UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE FILOSOFIA E CIÊNCIAS HUMANAS
CURSO DE BACHARELADO EM FILOSOFIA

RENATO REIS LEME

Programação como *forma de vida*: uma crítica ao representacionalismo na teoria da computação

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Filosofia

Orientador: Prof. Dr. Paulo F. E. Faria
Co-orientadora: Profª. Drª. Gisele Dalva Secco

Porto Alegre
2018

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Rui Vicente Oppermann
Vice-Reitora: Profª. Jane Fraga Tutikian
Pró-Reitor de Graduação: Prof. Wladimir Pinheiro do Nascimento
Chefe do Departamento de Filosofia: Profª. Sílvia Altmann
Coordenadora do Curso de Graduação em Filosofia: Profª. Ana Rieger Schmidt
Coordenador da Biblioteca de Ciências Sociais e Humanas: Vladimir Luciano Pinto

À memória de Alceu da Silva Leme, meu pai, que sempre acreditou no potencial transformador da educação e na importância da prática matemática no dia a dia.

AGRADECIMENTOS

Agradeço, em primeiro lugar, aos meus pais, a baiana dona Cida e o paranaense sr. Alceu, pela incansável luta empenhada na educação de seus filhos. Agradeço também aos meus irmãos, que, mesmo longe, sempre me incentivaram a superar meus próprios limites. À minha família, em especial tia Nena e tio Aloísio, que me receberam em São Leopoldo ao longo de todo o meu primeiro semestre do curso de Ciência da Computação na UFRGS, também agradeço pelo essencial apoio.

Agradeço também à todos os professores, colegas e amigos que pude conhecer ao longo do período em que passei na Universidade Federal do Rio Grande do Sul (UFRGS) e que influenciaram direta ou indiretamente na minha formação, tanto ao longo do curso de Bacharelado em Filosofia quanto no período em que cursei Ciência da Computação. Em especial, à professora Gisele Secco, que me mostrou ser possível olhar filosoficamente para a computação e acompanhou, corrigiu e guiou toda a produção deste trabalho; e ao professor Paulo Faria, pela inesgotável fonte de conhecimento, preciosas aulas e por ter acreditado nesta pesquisa desde o início.

Agradeço também aos professores Luciana Salete Buriol, Marcus Ritt e André Grahl Pereira pelo período em que pude viver minhas primeiras experiências como bolsista de Iniciação Científica, entre os anos de 2013 e 2015. Agradeço a todos os amigos do acolhedor grupo de pesquisa em algoritmos e otimização, também conhecido como "Lab 207", pelas divertidas e bem informadas discussões acerca dos mais variados assuntos.

Um dos valores que aprendi a cultivar no curso de filosofia é o da comunicação de ideias, de modo que eu não poderia deixar de agradecer aos meus caros amigos João Flach e Izabela Padilha pela amizade e pelo fértil período de produção intelectual em que estivemos imersos ao longo dos últimos anos. A nossa *esfera das aparências* sem dúvida influenciou este trabalho desde a sua concepção. À João, agradeço em particular por ter embarcado, junto a Renan Guarese, grande amigo que esteve comigo do início ao fim de nossos cursos de gradução, no projeto de programar o primeiro compilador para a linguagem de programação da teoria geral das operações de Wittgenstein. Agadeço também ao meu velho amigo Lucas Stefano Xavier de Sousa, pela amizade de longa data e pelas valiosas discussões ao longo dos cursos de verão no Instituto de Física Teórica da UNESP em 2014. Devo também um agradecimento especial à Lennon Macedo, rival no xadrez e crítico implacável de minha filosofia, por idealizar em 2015 o primeiro grupo de discussão filosófica do qual fiz parte, e a todos os demais integrantes do grupo por

acreditarem na importância da Filosofia. Agradeço também a todos os amigos e colegas que integraram o grupo de discussão da disciplina de *Seminário Avançado de Pesquisa* da professora Gisele Secco no período de $2017 - 2018$. Em particular, à Pedro Noguez, sem o qual eu teria grandes problemas para encontrar Kirchberg am Wechsel no interior da Áustria.

Gostaria de fazer um agradecimento especial a todos aqueles que, através do financiamento coletivo, tornaram possível a minha ida à 10^{a} Escola de Verão e ao 41^{o} Simpósio Internacional organizado pela *Austrian Ludwig Wittgenstein Society*. Sem dúvida nenhuma, a minha participação nesses eventos foi especialmente significativa na produção deste trabalho. O contato com pesquisadores do mundo inteiro, além de ampliar minha perspectiva acerca de diversos tópicos da filosofia da matemática de Wittgenstein, deu-me uma nova visão da pesquisa em Filosofia e me motivou ainda mais a seguir neste caminho.

Agradeço também à Anderson Nakano, tanto pelo cuidadoso trabalho que tem feito nos últimos anos acerca da filosofia da matemática do assim chamado "Wittgenstein intermediário" quanto pela generosa crítica à primeira versão do segundo capítulo desta monografia.

Agradeço ao professor Luiz Carlos Pereira, por ter proporcionado minha ida à 3^{a} Jornada de Mirantão, na qual tive a oportunidade de apresentar uma versão reduzida do argumento central do segundo capítulo desta monografia. O contato com grandes mentes da pesquisa em filosofia, matemática, lógica e computação, abriu meus olhos para uma série de novos e intrigantes aspectos relativos à filosofia da computação (ou filosofia da *informática*). Agradeço em especial à Bruno Lopes, pela generosa acolhida em Niterói na véspera de meu vôo de volta a Porto Alegre.

Por fim, quero agradecer à Jéssica Andrade, minha amada e maior fonte de inspiração, por partilhar comigo nossos melhores sonhos e pelo afetuoso apoio que me dedica.

RESUMO

Partiremos da hipótese de que a comunicação entre programador e máquina pode ser reduzida a duas grandes categorias: (A) comunicação direta ou (B) comunicação indireta. Diretamente, o programador comunica-se através da linguagem primitiva da máquina (seu conjunto de instrução); indiretamente, através de alguma linguagem de programação. Tradicionalmente, as linguagens de programação são analisadas segundo, de um lado, a sua sintaxe e, de outro, a sua semântica. A tradição *semanticista* das linguagens de programação sugere a necessidade de que a comunicação entre programador e máquina envolva a troca de alguma espécie de *conteúdo semântico*, apreensível pela máquina e compreensível pelo usuário. A tradição *operacional*, por sua vez, argumenta pela suficiência das *operações* definidas por cada linguagem e adequadamente traduzidas para cada arquitetura particular de computador. A partir da hipótese acima e da recente literatura acerca da comunicação indireta, a presente monografia buscará oferecer uma análise filosófica da computação com base no modelo proposto por Alan Turing em 1936. Como iremos argumentar, a partir da teoria das máquinas de Turing, podemos extrair as seguintes consequências: (i) o fenômeno da computação consiste no processo de manipulação simbólica; (ii) toda computação é efetuada por algum sujeito e (iii) esse sujeito é o computador. O conceito de linguagem de programação será então introduzido como *conjunto de instrução* e o de compilador como *transformador sintático* entre instruções de diferentes conjuntos. Como veremos, tais conceitualizações tornarão possível a dissolução do argumento da equivalência de entrada e saída, um dos argumentos com base em que o representacionalismo na computação, na reconstrução a ser oferecida, buscará se sustentar. Para tanto, argumentaremos pela existência de um elo conceitual entre o conceito de número do *Tractatus* e o conceito de número computável de Turing: em ambos, a noção metamatemática de *manipulação simbólica* é fundamental – no caso de Wittgenstein, é central como o método *da* matemática; no caso de Turing, central como o método do *computador*. Ao final, a crítica ao representacionalismo na computação será finalizada com base na perspectiva sugerida por Juliet Floyd em artigo recente: a saber, a partir da perspectiva do *uso* dos computadores. O conceito wittgensteiniano de *forma de vida* será então introduzido como um possível catalisador da multiplicidade essencial da computação, resolvendo o problema da ambiguidade proposto por Sprevak e concluindo a crítica.

Palavras-chave: TLP. Wittgenstein. filosofia da computação. filosofia da matemática.

linguagem. programação. Turing. teoria das máquinas. semântica formal. filosofia da mente.

Computer programming as *form of life*: a critique of representationalism in theory of computation

ABSTRACT

We will start from the hypothesis that the communication between programmer and machine can be reduced to two broad categories: (A) direct communication or (B) indirect communication. Directly, the programmer communicates through the primitive language of the machine (its instruction set); indirectly, through some programming language. Traditionally, programming languages are analyzed according to, on the one hand, their syntax, on the other hand, their semantics. The *semantical tradition* of programming languages suggests that the communication between programmer and machine necessarily involves the exchange of some kind of machine-readable semantic content that is both comprehensible to the user and to the machine. The *operational tradition*, in its turn, argues for the sufficiency of *operations* defined by each language and appropriately translated for each particular computer architecture. From the above hypothesis and the recent literature about indirect communication, this monograph will seek to offer a philosophical analysis of computation based on the model proposed by Alan Turing in his 1936. As we shall argue, from his *theory of machines*, we can draw the following consequences: (i) the phenomenon of computation consists in the process of symbolic manipulation; (ii) all computation is performed by some subject and (iii) this subject is the computer. The concept of programming language will then be introduced as *instruction set architecture* (ISA) and compiler as *sintatical transformer* between statements from different sets. As we shall see, such conceptualizations will make possible the dissolution of the input-output equivalence argument, one of the arguments on the basis that representationalism in theory of computation, in the reconstruction to be offered, will seek to sustain itself. For this, we will argue for the existence of a conceptual link between the concept of *Tractatus* number and Turing's concept of computable number: in both, the metamathematical notion of *symbolic manipulation* is fundamental - in the case of Wittgenstein, is central as *the method of mathematics*; in the case of Turing, central as the method of the *computers*. In the end, the critique of representationalism in computing will be finished based on the perspective suggested by Juliet Floyd in a recent article: namely, from the perspective of *use* of computers. The wittgensteinian concept of *form of life* will then be introduced as a possible catalyst for the essential multiplicity of computation, solving the problem of

ambiguity proposed by Sprevak, and concluding the critique.

Keywords: TLP, Wittgenstein, philosophy of computer science, philosophy of mathematics, language, programming, Turing, theory of machines, formal semantics, philosophy of mind.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TLP	*Tractatus Logico-Philosophicus*
ISA	Instruction Set Architecture
MTU	Máquina de Turing Universal
MT	Máquina de Turing

SUMÁRIO

1 INTRODUÇÃO

O cérebro eletrônico faz tudo
Faz tudo
Quase tudo
Quase tudo
Mas ele é mudo

Cérebro Eletrônico, Gilberto Gil

O assim chamado *cérebro eletrônico* foi, sem dúvida, uma das mais impactantes invenções tecnológicas da humanidade. De uns tempos pra cá, com a popularização dos dispositivos móveis, tais artefatos vêm se transformando em uma espécie de quinto membro do corpo humano, e não existem sinais de que essa tendência venha a diminuir. Pelo contrário: com as mais recentes possibilidades do mercado (como os cada vez mais viáveis óculos de realidade virtual, as novas e adaptadas arquiteturas, as "smart house" com sistema integrado de ação, etc), os computadores tendem a assumir papéis cada vez mais substanciais na vida humana. Embora seja argumentável que ainda estejamos apenas começando a aprender a conviver com esses "cérebros", não está no mérito desta monografia construir um juízo doxástico acerca dessa questão. Neste trabalho, estaremos preocupados em responder às seguintes questões: o que são, como funcionam e de que modo os cérebros eletrônicos se comunicam com outros cérebros, como, por exemplo, o nosso?

Seguramente, Gilberto Gil não quer dizer, no último verso da estrofe citada de sua célebre canção *Cérebro Eletrônico*, que computadores não sejam capazes de emitir sons. Sem dúvida, Gil sempre foi um músico habituado a apreciar melodias através dos mais variados meios físicos, e não teria motivo algum para não aceitar que computadores pudessem, também, ser capazes de emitir sons. Em particular, tais sons poderiam estar de tal modo encadeados que formassem frases completas da língua portuguesa. Sem dúvida, o autor da citada canção aceitaria que um computador suficientemente equipado pudesse (por que não?) ser tal qual um ser humano. Isto é, no que diz respeito ao fisicamente observável, é plenamente possível que se construa um computador (dotado de um "cérebro eletrônico") idêntico a um ser humano adulto médio.

Contudo, mesmo aceitando tudo isso, Gil ainda diria que tal cérebro eletrônico, apesar de tudo, é *mudo*. Mudo, na canção de Gilberto Gil, poderíamos extrapolar, exerce um papel lógico: é um termo técnico. Um termo técnico-filosófico: comprometeria seu

autor com a tese de que computadores possuem, ao menos, uma propriedade que os distingue fundamentalmente dos seres humanos (e vice-versa). Tal característica é, nomeadamente, a da *fala*. Mas se tais computadores são capazes de simular o comportamento dos seres humanos e, assim como eles, tomar parte em uma conversa *falada*, o que lhes faltaria? Se a resposta for "faltaria *alguma* coisa", como Gil parece querer nos sugerir, então isso implicaria o anti-reducionismo computacional da mente humana.

A presente monografia, contudo, não pretende oferecer uma crítica ao reducionismo mental. Trata-se de uma crítica a uma de suas vertentes: o representacionalismo semântico. No passado, a tese representacionalista teve grande influência como sustentáculo do funcionalismo. O que mais tarde viria a se transformar na hipótese *computacionalista* da mente humana hoje mantém-se vivo nos debates de ontologia das linguagens de programação. O assim denominado *representacionalismo* na computação consiste na seguinte tese **R**

R Computação envolve essencialmente conteúdo representacional.

Para compreender o conteúdo afirmado por esta tese, devemos ter em mente, em primeiro lugar, o que seja *computação*. Tendo em vista apresentar este difícil conceito, a presente monografia buscará oferecer uma perspectiva *top-down* de suas notas características: para tanto, iniciaremos nossa investigação pela computação *qua* fenômeno causal observável, como será melhor descrito na próxima seção. Em seguida, a *teoria das máquinas* de Turing será introduzida e o conceito de *computador* formalizado. O modelo proposto para a computação conduzir-nos-á, por fim, à terceira questão central desta monografia, a saber: qual a natureza da comunicação entre ser humano e máquina? Para responder a esta pergunta, partiremos de um caso paradigmático: a programação de computadores. Nesse contexto, a tese representacionalista será então reintroduzida como uma resposta possível para a pergunta acerca da natureza de tal comunicação.

1.1 O que é *computação*?

O estudo da *computação*, como fenômeno, caracteriza-se por seu aspecto altamente interdisciplinar. A *ciência* da computação, por exemplo, é notoriamente matemática: dentro de seu escopo, pode-se formular, testar e provar teoremas, calcular complexidade e correção de algoritmos, oferecer soluções ótimas para problemas específicos (como para o problema de encontrar um algoritmo de solução ótima com reduzida árvore

de busca para o jogo do *Sokoban* (LEME et al., 2015)) etc.

A *engenharia* da computação, por sua vez, como o nome sugere, aborda o fenômeno desde uma perspectiva distinta. Outras áreas de estudo abordam a computação de modo mais indireto: como é o caso da área de atuação daqueles que se dedicam ao desenvolvimento de linguagens de programação, ou ao desenvolvimento de jogos, ou até mesmo à física dos cabos de fibra óptica. A *ética* da computação, como colocado em uma introdutória incursão à *filosofia* da computação (TURNER; ANGIUS, 2017), aborda questões éticas relativas ao fenômeno da computação: não questões como "seria ético ensinar os computadores a amar?", mas sim questões que refletem problemas reais, como as relativas aos direitos de patente e de reprodução de conteúdo, direito a privacidade, direitos do consumidor, etc, quando envolvendo dispositivos computacionais.

A presente monografia enquadra-se, dentro do panorama brevemente delineado acima, como uma investigação de cunho *filosófico* do fenômeno da computação. Estará interessada em responder à questão "o que é a computação?" desde o ponto de vista de suas características mais gerais. Nesse sentido, nossa investigação poderia ser caracterizada como uma espécie de metafísica da computação, ainda que o termo pudesse ser mal recebido. Como, de fato, nosso maior objeto de investigação serão as linguagens de programação, talvez esta monografia pudesse ser mais adequadamente apresentada como filosofia das linguagens de programação. Nossa maior preocupação, de todo modo, será a adequação lógica da apresentação: esperamos, com isso, no mínimo oferecer uma narrativa coerente sobre algum aspecto da realidade.

Para avançar nossa investigação metafísica, partiremos da premissa de que é apenas mediante os *computadores* que o fenômeno da computação pode ser objeto de investigação de qualquer uma dessas abordagens: o fenômeno da computação é capturado (bem ou mal) pelo conceito de *computador*, que nada mais é do que o *agente* da computação, na medida em que o fenômeno da computação decorre da lógica de ação do computador. Em outras palavras, a contrapartida empírica da computação é o computador e é este o genuíno objeto direto da oração "a XYZ da computação *investiga* ..."[1].

[1]Talvez seja importante observar que, desde esta perspectiva, não pretendo reduzir a plural variedade de áreas de pesquisa em computação ao estudo de um único objeto, mas sim operar um corte como o sugerido por Ockham. Eu não poderia, por exemplo, caracterizar computadores como os únicos agentes da computação se tal corte não fosse realizado, pois, nesse caso, os supercomputadores seriam um possível contra-exemplo hipotético.

1.2 A teoria das máquinas de Alan Turing

Amplamente falando, computadores são os agentes da computação. Um caixa eletrônico disponibiliza notas mediante a solicitação do cliente: o caixa é o agente das computações necessárias para que o *input* (solicitação do cliente) resulte no *output* (notas solicitadas).

Contudo, na melhor das hipóteses, a caracterização acima do que seja um computador é *filosófica*: não é suficiente, portanto, para uma investigação científica do fenômeno da computação. De fato, nas origens da computação como disciplina científica está o artigo de Alan Turing 'On computable numbers, with an application to the *Entscheidungsproblem*', publicado em 1936 para demonstrar a impossibilidade lógica da existência de um algoritmo suficientemente genérico que respondesse 'Sim' ou 'Não' para uma pergunta matemática qualquer, como "Dados dois números inteiros (x e y), x é divisível por y?"[2]. Nesse artigo, Turing torna palatável o conceito de computador: a por ele denominada "teoria das máquinas" oferecerá uma série de definições precisamente delimitadas para o conceito de computador (máquina) e será exposta na seção a seguir.

1.2.1 As diversas *máquinas* de Turing

Em seu seminal 'On computable numbers...', Turing avalia o fenômeno da computação operada por seres humanos. Imagina o cenário de homens trabalhando com o cálculo de grandes equações e se questiona pelo mínimo que tais operários necessitariam não só para poder trabalhar como também para poder fazer uma pausa (para tomar um café, sair para o fim de semana ou até mesmo para umas férias) e retomar novamente seu trabalho, exatamente de onde pararam, e segundo os mesmos procedimentos de sempre[3].

> 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, ..., q_R, as quais chamaremos de "configurações-*m*". A máquina é equipada com uma "fita" (o análogo do papel) que passa por ela e é dividida em seções (chamadas de "quadrados"), cada uma capaz de armazenar um "símbolo". A qualquer momento há apenas um quadrado, digamos o r-ésimo, armazenando o símbolo $\mathfrak{S}(r)$ que está "na máquina". Podemos chamar este quadrado de

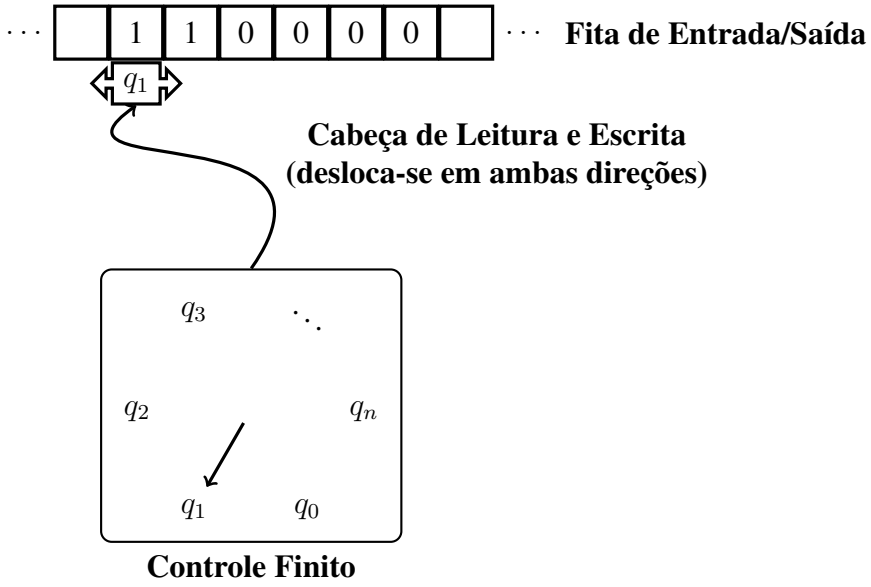
[2] A formulação precisa do problema se deve a Hilbert e Ackermman, que em 1928 definiram o desafio como o problema de "encontrar um procedimento para demonstrar se uma dada fórmula no cálculo de proposições de primeira ordem era válida ou não" (DIVERIO; MENEZES, 2009).

[3] Nesse artigo, máquinas são computadores e computadores são seres humanos calculando. Isso se deve ao fato, digno de nota, de que na época em que Turing escreveu seu artigo era comum a demanda por *computadores* no mercado de trabalho da Inglaterra: tais verdadeiros operários do cálculo dedicavam horas de seu dia ao trabalho de *computar*.

> "quadrado examinado". O símbolo no quadrado examinado pode ser chamado de "símbolo examinado".

Turing inicia o paragráfo com a sua célebre analogia entre um "homem no processo de computar um número real" e uma "máquina ... capaz apenas de um número finito de condições [instruções]". A julgar pelo texto, este é o paradigma que ele espera que o leitor tenha em mente durante a leitura do artigo. O matemático, em sua análise, destaca três componentes da arquitetura desta máquina, a saber: **(i)** fita **(ii)** configurações-*m* **(iii)** dispositivo de leitura e escrita

Figura 1.1: Esquematização dos componentes básicos de uma máquina de Turing.



Fonte: Retirado de <http://www.texample.net/tikz/examples/turing-machine-2/>.

Na abstração proposta por Turing, a fita é o meio físico da entrada e da saída das computações realizadas pela máquina, que opera segundo as assim chamadas configurações-*m*. O dispositivo de leitura e escrita é um elemento essencial da teoria, pois é através dele que o computador interage com a fita e, consequentemente, com o meio físico.

> O "símbolo examinado" é o único do qual a máquina está, por assim dizer, "diretamente consciente". Contudo, alterando sua configuração-*m*, a máquina pode efetivamente lembrar alguns dos símbolos que foram "vistos" (examinados) previamente.

É importante, neste ponto, que não se deixe levar pelo vocabulário psicológico empregrado por Turing. Note-se que, ele próprio, diversas vezes lança mão das aspas para indicar que não está se comprometendo com o sentido forte do termo, como em "diretamente consciente" e "visto", na passagem citada acima. Como observou Floyd, Turing não teria alcançado seu objetivo de obter uma prova negativa para o *Entscheidungsproblem* se suas afirmações dependessem de uma teoria psicológica de fundo.

> O comportamento possível da máquina a qualquer momento é determinado pela configuração-*m* q_n e pelo símbolo examinado $\mathfrak{S}(r)$. Este par q_n, $\mathfrak{S}(r)$ será chamado de "configuração": assim, a configuração determina o comportamento possível da máquina.

Na sequência do parágrafo, Turing introduz mais um elemento de extrema importância para a sua análise. Computadores, em sua acepção, possuem sua ação inteiramente determinada pelas configuração-*m* em função do símbolo examinado, isto é, de sua interação com o meio.

> Em algumas das configurações em que o quadrado examinado está em branco (isto é, ele não armazena nenhum símbolo) a máquina escreve um novo símbolo no quadrado examinado: em outras configurações ela apaga o símbolo examinado. A máquina pode também mudar o quadrado que está sendo examinado, mas apenas deslocando-o para o lado direito ou esquerdo. Além dessas operações, a configuração-*m* pode ser alterada.

A partir deste ponto, Turing introduz as operações possíveis para a máquina descrita, a saber: se o quadrado examinado está em branco, ela pode escrever algum símbolo; senão, pode apagar o símbolo lido. Além disso, a máquina é capaz de deslocar-se para a esquerda e para a direita na fita, mudando, assim, o quadrado examinado. A possibilidade de se alterar configurações-*m*, mencionada ao final desta passagem, sugere o alcance potencial do formalismo proposto: é através da alteração na configuração-*m* que se produz novas máquinas.

> Alguns dos símbolos escritos irão formar a sequência de figuras que é o decimal do número real que está sendo computado. Os outros são meras anotações para "auxiliar a memória". Apenas estas anotações estarão sujeitas a serem apagadas.

Com que satisfação Turing não teria vivenciado o surgimento dos primeiros *videogames*? A verdade é que tal invenção só foi possível porque se percebeu que a mesma "sequência de figuras" que codificam o "número real que está sendo computado" poderia codificar uma série de outras coisas, dentre as quais estão os (hoje em dia) essenciais *pixels* de uma tela de computador. Neste momento, porém, Turing tem em vista apenas os números computáveis: além do resultado mesmo da computação, diz-nos ele, é permitido à máquina outros caracteres constitutivos de "auxiliares" para a memória: isso porque, frequentemente, para realizar computações complexas, é conveniente que nos guiemos por marcas de nossos passos anteriores.

> Argumento que estas operações incluem todas aquelas que são usadas na computação de um número. A defesa desse argumento será mais fácil quando a *teoria das máquinas* for familiar ao leitor. Na próxima seção, portanto, procedo no desenvolvimento da teoria e assumo que se entende o que "máquina", "fita", "examinado", etc, significam.
>
> – 'On computable numbers...', A. Turing, pp. $231 - 232$[4]

[4]Tradução e grifos meus.

Não há nenhuma referência externa a essa ocorrência de *teoria das máquinas*. Isso sugere que tal teoria é parte do que Turing propõe em seu artigo: a saber, uma teoria da computação que estava sendo pensada pelo próprio autor, uma criação sua. No centro dessa teoria estava a analogia entre máquinas (genuínos agentes de pensamento simbólico) e seres humanos (capazes de simular o comportamento de qualquer máquina) – em outras palavras, a teoria das máquinas de Turing preserva, em sentido preciso, a complexa diversidade dos aspectos que nos fazem distintos dos computadores *em princípio*. Uma ideia que Gilberto Gil, penso eu, certamente apreciaria.

Após introduzir os elementos básicos de suas máquinas, com muita justiça Turing supõe que seu leitor, a partir daquele momento, saberá do que ele estará falando quando falar de máquinas. Seguindo na exposição de sua teoria, o próximo passo será expor algumas *espécies* de máquinas, as quais ele denomina: (**I**) máquinas automáticas, (**II**) máquinas de computação, (**III**) máquinas circulares e (**IV**) máquinas livres de círculo.

> **Máquinas automáticas** Se cada estágio de movimento de uma máquina (no sentido de §1) está completamente determinado pela configuração, chamaremos a máquina de "máquina automática" (ou uma máquina-*a*).
>
> **Máquinas de computação** Se uma máquina-*a* escreve dois tipos de símbolos, dos quais o primeiro tipo (chamado de figuras) consiste inteiramente de 0 e 1 (e os outros sendo chamados de símbolos de segundo tipo), então a máquina será chamada de uma máquina de computação. Se se fornece uma fita branca para a máquina e a coloca em movimento, começando da configuração-*m* inicial correta, a subsequência de símbolos de primeito tipo que são escritos por ela será chamada de *sequência computada pela máquina*. O número real cuja expressão como decimal binário é obtido prefaceando-se esta sequência por um ponto decimal é chamado de *número computado pela máquina*.
>
> A qualquer estágio de movimento da máquina, o número do quadrado escaneado, a sequência completa de todos os símbolos da fita, e a configuração-*m* descreverão a *configuração completa* de cada estágio. As mudanças da máquina e da fita entre configurações completas sucessivas será chamada de *movimentos* da máquina.
>
> **Máquinas circulares e livres de círculo** Se uma máquina de computação nunca escreve mais do que um número finito de símbolos do primeiro tipo, ela será chamada de *circular*. Do contrário, será chamada de *livre de círculo*.
>
> Uma máquina será circular se atingir uma configuração a partir da qual não existe mais movimento possível, ou se continuar executando, e possivelmente escrevendo símbolos do segundo tipo, mas não podendo mais escrever qualquer símbolo do primeiro tipo.

A teoria das máquinas é uma contribuição ainda bastante rudimentar à teoria da computação de um modo geral, mas o mais importante é que pela primeira vez alguém, como coloca Floyd, *domesticou* a idealizada máquina que resolveria o problema de Hilbert em termos de uma teoria simples porém extremamente frutífera do que seja um computador ou, nos termos de Turing, uma *máquina de computação*.

Turing diria, portanto, que um exemplo genuíno de *máquina de computação* poderia ser o funcionário de um escritório responsável pelo cômputo das despesas do mês. Ou o gerente de um supermercado na função de calcular a balança comercial depois de um dia de expediente. O importante é que a cada uma dessas máquinas, para realizar aquilo que lhes cabe enquanto tais, bastaria um dispositivo de leitura e escrita, uma fita e configurações-*m*.

1.2.2 Máquina de computação universal

Computadores como nossos smartphones, notebooks, servidores web e etc, são os que vieram a ser mais tarde nomeados de *computadores de propósito geral*. Tais computadores são assim chamados porque são *programáveis*. Mais do que isso, poder-se-ia argumentar que tais computadores são capazes de computar *quaisquer* algoritmos computáveis[5]: bastaria uma programação adequada.

Contemporaneamente, tal propriedade peculiar a esses dispositivos computacionais se refletiu na explosão de linguagens de programação ocorrida na segunda metade do século XX. Turing anteviu a decorrência lógica dessa possibilidade a partir de suas próprias máquinas e denominou tais dispositivos por *máquinas de computação universais* (§6, p. 241).

> É possível inventar uma única máquina que pode ser usada para computar qualquer sequência computável. Se se fornece a essa máquina U uma fita no começo da qual está escrita a descrição standard de alguma máquina de computação M, então U irá computar a mesma sequência que M.

Boolos introduz o conceito da seguinte maneira (BOOLOS; BURGESS; JEFFREY, 2012)

> Uma máquina de Turing para computar uma função universal é denominada uma máquina de Turing universal.

Sendo uma função universal F definida como

> ... função recursiva $(n+1)$-ária F com a propriedade de que para toda função recursiva n-ária f há um m tal que
>
> $$f(x_1, ..., x_n) = F(m, x_1, ..., x_n)$$

[5]Tal é o que parece sugerir Davis (DAVIS, 2004), embora a discussão esteja, mais do que nunca, em aberto. Computadores como nossos notebooks, na medida em que possuem um conjunto mínimo de instruções realizáveis, pode-se provar que são turing-completos (isto é, computam as mesmas funções de que uma máquina de Turing é capaz): a hipótese da hipercomputação nos convida a considerar a possibilidade de que existam (no sentido de que podem ser construídos) computadores capazes de computar ao menos uma função que não seja turing-computável.

A definição de função universal, portanto, ao incluir no domínio de F os objetos m, incorpora todo o domínio de variação das funções f no domínio de uma única grande função. Com isso é possível? A resposta, que Turing compreendeu muito bem, está na peculiar característica que tais funções n-árias f possuem, a saber, a de serem *codificáveis*.

A existência de um procedimento algorítmico de codificação de cada f implica a existência de um algoritmo de decodificação. O papel principal da função F, portanto, está em sua capacidade de decodificar m nos termos de f e operar com os argumentos $x_1, ..., x_n$ da mesma forma como f operaria. A descoberta de F subjaz a descoberta dos modernos computadores de propósito geral, e ambas encontram raiz na teoria das máquinas de Alan Turing.

1.2.3 Mecanismo de computação automática

Ao final de 1945 (9 anos após a publicação de 'On computable numbers', portanto) Turing chefeou, no National Physical Laboratory (NPL) em Londres, o projeto de viabilizar o assim chamado Automatic Computing Engine (*Mecanismo de Computação Automática*), projeto de computador de programa armazenado idealizado por Turing e extremamente sofisticado que, por motivos financeiros, não pôde ser concluído a tempo (uma versão mais simples, no entanto, foi implementada após a saída do matemático do projeto).

Com a expansão da indústria digital do pós-*Segunda Guerra Mundial*, os computadores (conceitualmente viabilizados primeiramente pelo trabalho de Turing) possuíam o cenário econômico ideal para surgirem como produto de uma cadeia de produção. A chamada "corrida para construir o primeiro computador digital eletrônico de programa armazenado do mundo", cujo vencedor foi um computador denominado *Manchester Baby* em 21 de junho de 1948 ("Manchester" por ter sido criado na Universidade de Manchester e "baby" por tratar-se de um computador extremamente simples). Segundo nos conta Copeland (COPELAND, 2005),

> As its name implies, the Baby was a very small computer, and the news that it had run what was only a tiny program– just 17 instructions long–for a mathematically trivial task was 'greeted with hilarity' by Turing's group.

A hilariedade com a qual o grupo de Turing recebeu a notícia de que o computador de Manchester havia computado apenas uma simples tarefa matemática composta por 17 instruções certamente não foi gratuita. Não que o computador de Manchester não fosse capaz de realizar computações complexas, mas sim que, sendo teoricamente capaz de

computar funções computáveis com grandes entradas e muita computação, a arquitetura do projeto tornava tal tarefa fisicamente inviável: era preciso *muito* para produzir *pouco*.

Turing e sua equipe, de fato, trabalhavam em um computador muito mais sofisticado que o de Manchester, como nos conta Copeland. Infelizmente o mecanismo de Turing não se viabilizou em sua inteireza. Fica-nos, contudo, a memória de um dos últimos frutos de sua teoria das máquinas.

1.2.4 Turing e a computação simbólica

Gostaria de sugerir, como conclusão desta seção, que a teoria das máquinas de Turing se comprometia com a seguinte tese filosófica **S**

Tese S Computação é manipulação regrada de símbolos.

Em outras palavras, o que para Turing talvez fosse questão de definição (como nos diz, uma máquina de computação é uma máquina tal que escreve e lê *símbolos*) de fato revela-nos uma certa postura filosófica. Tal postura filosófica, como pretendo argumentar no próximo capítulo, estava também presente na definição de Wittgenstein para o conceito de número no aforismo 6.02 do *Tractatus Logico-Philosophicus*.

Por ora, porém, gostaria de voltar ao início deste capítulo, quando mencionei a tese representacionalista **R**. Como ficará mais claro adiante, tal tese opõe-se à tese **S** na medida em que introduz a necessidade de que os símbolos manipulados pela máquina estejam em alguma relação de referência por representação com objetos de algum domínio externo ao próprio processo de manipulação simbólica. A computação como manipulação de símbolos passaria a ser, como nos dirá Sprevak, *manipulação de representações*.

Se a computação consiste na manipulação de representações, então a semântica das linguagens de programação deve ser capaz de conferir *conteúdo* a cada uma de suas instruções possível. O próprio Sprevak não é explícito com relação a noção de *representação* com a qual se compromete, mas, como irei sugerir, tal noção está fortemente relacionada com a hipótese de que a mente humana opera segundo mecanismos de manipulação de representações. A noção de representação defendida por Sprevak seria, portanto, análoga à noção de representação como estado mental, e o comprometeria com uma teoria das linguagens de máquina incompatível com o paradigma de Turing.

1.3 O que são linguagens de programação?

Em uma primeira aproximação, podemos dizer que *linguagens de programação* são as ferramentas através das quais escrevemos *programas de computador*. Programas de computador, por sua vez, são conjuntos de instruções. A questão de saber o que são linguagens de programação, poderia, portanto, ser reformulada da seguinte maneira: que tipo de construções são capazes de simbolizar uma instrução? Ora, sabemos que um dos meios pelos quais podemos explicitar instruções é a linguagem natural. Poderíamos, nesse sentido, escrever o seguinte programa

Instrução 1 Ir até o endereço Rua São Manoel, 230.

Instrução 2 Solicitar 5 pães ao primeiro balconista disponível.

Instrução 3 Ir até o endereço de casa.

... e fazer de nosso sobrinho um computador a serviço do café da manhã.

A teoria das máquinas de Turing parte do fato de que o fenômeno da computação já está, de uma forma ou de outra, embutido em nosso mundo como uma espécie de processo causal tal qual qualquer outro. Ao formalizar o conceito de computador, Turing constrói um modelo matemático para este fenômeno particular. Acontece que, no contexto da teoria das máquinas, não apenas as instruções são símbolos como também todos os possíveis conjuntos de instruções são símbolos, denominados por Turing de *descrições standard*. Pode-se construir uma máquina de Turing para decodificar cada uma das descrições standard possíveis unicamente através de manipulações simbólicas. Assim, pode-se construir uma única máquina de Turing capaz de decodificar qualquer descrição standard: tal é o princípio da máquina Universal que atesta o fato de que, no escopo de teoria de Turing, o método de manipulação simbólica atua universalmente.

Neste ponto da monografia, isto é, partindo do arcabouço conceitual oferecido pela teoria das máquinas de Turing, estamos em plenas condições de dar um passo além em nossa análise da computação. Já encontramos uma saída para a questão "o que é computação?" através de uma distinção entre o fenômeno da computação e o seu agente de causa, com o que associamos o conceito de *computador*: objeto de investigação central da teoria da computação. Desde a perspectiva da teoria de Turing, computadores são agentes causais. A computação é um processo causal e, como tal, assume diversas formas: um programa de computador capaz de reproduzir arquivos *.mp3 poderia reproduzir uma

cópia digital do disco *Transformer*, de Lou Reed – nesse caso, o processo da *computação* consistiria das manipulações simbólicas internas ao programa, e o produto dessa mesma computação seriam as vibrações sonoras propagadas pelos meios apropriados.

Contudo, como podemos dizer para um computador o que ele deve fazer? Afinal de contas, é na capacidade de se comunicar com o usuário, característica peculiar aos computadores modernos, que reside o supra-sumo de sua utilidade. Tal problema é central nos estudos da relação entre homem e máquina, e consiste em uma das preocupações centrais da área de pesquisa em *linguagens de programação*.

Estritamente falando, a linguagem de um computador é um conjunto de palavras (expressões bem formadas). Tal conjunto poderia ser determinado por alguma espécie de gramática, mas também poderia ser dado por outros meios, como, por exemplo, uma tabela. Considere a tabela 1.1 a seguir, descritiva das ações do computador Neander. O Neander é uma das máquinas hipotéticas de Raul Fernando Weber (WEBER; PôRTO, 1998), ex-professor do Instituto de Informática da UFRGS e um dos pioneiros na área de pesquisa em arquiteturas de computador no Brasil.

Tabela 1.1: A "linguagem" do Neander: uma tabela de 11 instruções.

código	instrução	comentário
0000	NOP	nenhuma operação
0001	STA end	armazena o acumulador
0010	LDA end	carrega o acumulador
0011	ADD end	soma
0100	OR end	"ou"lógico
0101	AND end	"e"lógico
0110	NOT	inverte (complementa) acumulador
1000	JMP end	desvio incondicional
1001	JN end	desvio condicional
1010	JZ end	desvio condicional
1111	HLT	término de execução

Extraído do livro *Fundamentos de arquitetura de computadores*, de Raul Weber.

Um algoritmo de soma simples formulado nos termos da tabela poderia ser como o seguinte

Instrução 1 : **LDA** 0001

Instrução 2 : **ADD** 0010

Instrução 3 : **STA** 0011

Exatamente os mesmos comandos estão codificados, no dialeto do Neander, pela

palavra $0010\,0001\,0011\,0010\,0001\,0011$, gerada pela manipulação dos símbolos LDA, ADD e STA conforme a tabela 1.1.

As máquinas de Weber são descritas como hipotéticas, mas é importante observar que tais máquinas são hipotéticas em um sentido muito diferente daquele pelo qual dizemos que hipercomputadores são hipotéticos. O Neander é plenamente implementável, e, além dos simuladores criados pelo próprio Weber, outras implementações já foram propostas e desenvolvidas[6]. As máquinas de Weber são hipotéticas por motivos puramente contingentes: a saber, porque não requerem, dado seu propósito didático, implementação física – são suficientes os *simuladores* de sua arquitetura.

Chamei a atenção para o Neander porque, mesmo sendo extremamente simples e de fácil apresentação, o seu princípio de ação é o mesmo que o de computadores complexos. A *tabela de instruções* do Neander pode ser concebida como sua linguagem de programação *primitiva*. É através de tal linguagem que um usuário do Neander pode programar as rotinas de seu interesse, como a rotina escrita acima para o cômputo da soma de dois inteiros quaisquer de até 8 dígitos binários. A operação é, evidentemente, dependente do computador: as instruções só fazem sentido para o Neander e suas simulações (uma das consequências da teoria das máquinas de Turing é que um ser humano com uma descrição da lógica de funcionamento do Neander poderia, em princípio, simulá-lo inteiramente).

No entanto, poder-se-ia argumentar: se não há nada além (se não há uma representação de *como* operar com as instruções), então a tabela não seria suficiente para desencadear qualquer processo causal, e, portanto, as instruções descritas pelo algoritmo não seriam suficientes para produzir o fenômeno da computação. Tal visão é sugerida, por exemplo, por Turner

> Programas possuem seus significados dados pelos significados de suas construções internas, e, geralmente, a semântica deve preservar os significados das construções para além dos programas.
>
> – R. Turner, 'Programming languages as technical artefacts', p. 3.

O papel da semântica na programação, segundo essa visão, é essencialmente similar ao papel da semântica na análise das linguagens naturais: a saber, o papel de oferecer o significado termo a termo das construções frasais com sentido. Uma vez que representações são essenciais, a linguagem através da qual nos comunicamos com os computadores deve possuir uma semântica que confira conteúdo para cada uma de suas fórmulas bem

[6]Um exemplo é o projeto HidraCPP do grupo PET-Computação da UFRGS, do qual fiz parte durante o período de 2013-2014, e cujo objetivo é oferecer um único simulador para todas as máquinas de Weber. Tal projeto pode hoje ser acessado através da página <https://github.com/petcomputacaoufrgs/hidracpp>.

formadas.

Como coloca Rescorla (RESCORLA, 2013) (tradução e grifos meus)

> Alguns filósofos sustentam que um sistema físico implementa uma computação apenas se o sistema tiver propriedades representacionais (Crane [1990]; Fodor [1998], p. 10; Ladyman [2009b]; Sprevak [2010]). Nas palavras de Ladyman, "para estados físicos contarem como estados computacionais, eles devem ser genuinamente representacionais" ([2009b], p. 382). Da mesma forma, Sprevak ([2010], p. 260) afirma que "o apelo ao conteúdo representacional é inescapável ao atribuir computações a sistemas físicos". Chame isso de *visão semântica* da implementação computacional. Na visão semântica, todos os sistemas físicos computacionais possuem propriedades semânticas ou representacionais.

Contra essa visão, duas críticas se destacam na bibliografia e nos remetem aos trabalhos de Frances Egan (EGAN, 1992) e, mais recentemente, de Gualtiero Piccinini. Ambos autores buscaram criticar o representacionalismo na computação: de um lado, Egan rejeitou a necessidade de conteúdos semânticos na computação ao enfatizar o caráter matemático dos computadores, os quais conceitualizou como objetos puramente matemáticos; de outro, Piccinini (PICCININI, 2008), em uma crítica mais bem acabada, atacou o representacionalismo pelo esclarecimento do mecanismo através do qual processos de computação são conduzidos: como argumentou em seu artigo de 2008, em um computador, "propriedades funcionais são especificadas por uma explicação mecanicista sem apelar para qualquer propriedade semântica".

Na presente monografia, estarei interessado em defender a posição de que a programação de computadores é um sub-fenômeno da computação e, como tal, também se reduz a manipulações regradas de símbolo. Para tanto, irei propor a seguinte divisão: a comunicação humano-máquina pode ser de dois tipos: (A) comunicação direta; (B) comunicação indireta.

1.3.1 Comunicação direta e indireta

O fenômeno da comunicação é assombrosamente amplo. Contudo, no que diz respeito à matemática da comunicação, como Shannon faz questão de enfatizar nas primeiras páginas de seu clássico artigo (SHANNON, 1948), de nada importam considerações acerca de aspectos semânticos. A análise deve não só dispensá-las como deve buscar evitá-las. Nesta seção, buscarei explicar os mecanismos de comunicação do computador partindo do pressuposto básico de que a espécie de comunicação de que tais computadores são capazes não envolve nenhuma capacidade de *representar* a partir dos sinais recebi-

dos como entrada. Em outras palavras, a saída do computador resulta de manipulações simbólicas desprovidas de qualquer significado para a máquina que computa[7].

Partindo da teoria das máquinas de Turing, temos em mãos as características centrais do que seja um computador. Com base nisso, gostaria de me encaminhar para uma resposta à terceira e última questão central desta monografia, a saber: qual a natureza do fenômeno de comunicação existente entre ser humano e máquina? Interessar-nos-á saber, para tanto, se o tipo de comunicação estabelecida entre seres humanos e máquinas é do mesmo tipo daquela que máquinas estabelecem entre si (pode-se imaginar um programa de computador acionando outro[8]).

Para tanto, gostaria de traçar a seguinte distinção: a *comunicação direta* com um computador realiza-se por meio de suas próprias instruções, isto é, a entrada recebida consiste de um conjunto de instruções do próprio computador; a *comunicação indireta*, por sua vez, ocorre mediante a execução de algum programa do computador, capaz de transformar a entrada em instruções do computador alvo.

Comunicação direta Um exemplo de comunicação direta poderia ser pensado a partir da máquina do professor Weber: todo algoritmo escrito na linguagem do Neander pode ser executado diretamente por qualquer uma de suas implementações. Isso porque a lógica interna do Neander é tal que existe uma correspondência direta com a sua tabela de instruções: toda entrada que consista de instruções do Neander é simbolicamente idêntica a uma fórmula bem formada (palavra) da linguagem deste computador particular.

Comunicação indireta Diremos que a linguagem do Neander é uma linguagem primitiva da máquina para diferenciá-la das demais linguagens, que são construídas independentemente dos computadores, mas que dependem deles para que possuam qualquer espécie de eficácia intramundana[9].

Poderíamos implementar um compilador para a linguagem C++ com as instruções

[7]Um paralelo poderia ser traçado com o famoso experimento mental de Searle: o quarto chinês é um argumento contra o funcionalismo que enfatiza justamente a dispensabilidade do critério de compreensão para a computação de saídas complexas a partir de entradas (por definição) incompreensíveis para o computador.

[8]Em programação, é comum o uso de bibliotecas para a organização do código. A chamada de subrotinas constitui um caso paradigmático de comunicação entre diferentes programas de computador.

[9]Acredito que a principal fonte de enriquecimento da linguagem humana consista das relações de intersubjetividade estabelecidas nos ambientes de desenvolvimento do falante. A ideia de uma eficácia intramundana só pôde figurar nesta seção graças às valiosas aulas de Filosofia Política com os professsores Pertille e Felipe Gonçalves. Dessas aulas, guardo a lição de que a eficácia de uma teoria se mede pelo seu grau de penetrabilidade no mundo real, cotidiano: argumento que o mesmo critério, tão necessário para teorias morais, também poderia ser aplicado para diferenciar teorias da computação.

do Neander. De fato, convém pensar o conjunto de instuções de um computador como um conjunto de blocos, a partir dos quais poder-se-ia construir edficícios das mais variadas complexidades. Gostaria de argumentar que o mesmo se aplica para as fórmulas bem formadas de uma linguagem de programação: tais fórmulas são o equivalente ao edifício construído com os blocos, os quais, por sua vez, são os equivalentes lógicos das instruções que constituem o conjunto de instrução da linguagem de programação em particular.

1.3.2 O representacionalismo na computação

Na terceira seção de seu artigo (SPREVAK, 2010), Sprevak oferece três argumentos em favor da tese representacionalista **R**. No que se segue, introduzo uma nomenclatura para cada um dos argumentos.

O argumento dos casos paradigmáticos Diversos modelos paradigmáticos (como o de Turing) envolvem representação (pp.21-22)

O argumento da equivalência de entrada e saída A equivalência de entrada e saída de dois sistemas computacionais só pode ser garantida pelo acréscimo da dimensão das representações (pp. 22–24)

O argumento das distinções básicas A diferença entre duas funções de verdade não pode ser capturada, em uma implementação, unicamente pela tabela de verdade: em particular, OR(xRb):VVVF pode ser AND(xRb):VFFF em duas implementações fisicamente idênticas; é o acréscimo da representação que resolve a ambiguidade a nível operacional (pp. 24–26).

Ao final deste capítulo, espero ter oferecido razões suficientes para rejeitarmos seu primeiro argumento. No segundo capítulo desta monografia, me dedicarei a recusar o segundo deles. No terceiro, por fim, oferecerei uma crítica ao seu terceiro argumento no contexto de uma discussão mais ampla acerca da programação de computadores.

1.3.2.1 O argumento dos casos paradigmáticos

> Muitos casos paradigmáticos de computação envolvem representação. Por exemplo, o funcionário descuidado de Turing, que efetua computações com as mãos, efetua um *mapeamento entre representações*. O funcionário mapeia representações (marcas de tinta no papel) para outras representações (outras marcas de tinta no papel). [10]

[10] Tradução e grifos meus.

Se se define representação como mera *marca de tinta no papel*, Sprevak afima uma verdade acerca do formalismo de Turing. Contudo, o próprio Sprevak não oferece nenhuma razão para que aceitemos semelhante reducionismo. De fato, uma das consequências imediatas seria o colapso da categoria de *representação* com a categoria de *sinal*, sob pena de que todo sinal, em princípio, *representaria* alguma coisa. Tal trivialização do conceito de representação, contudo, não é sequer cogitada pelo próprio Sprevak, que julga uma espécie de consenso ou obviedade o fato de que se possa equacionar *marcas de tinta no papel* com *representação* de maneira tão direta.

De fato, se estivemos corretos até aqui em nossa apresentação da teoria das máquinas de Turing, o primeiro argumento de Sprevak sequer sai do chão: a principal premissa do matemático britânico, como vimos, é justamente que a manipulação operada pelo computador é puramente simbólica. O "funcionário descuidado" de Turing não descuida-se por acaso: no que diz respeito à computação de um algoritmo, isto é, do ponto de vista das condições suficientes e necessárias para que, do algoritmo W, resulte a computação Y, as *marcas de tinta no papel* nada representam.

1.3.2.2 O argumento da equivalência de entrada e saída

> Imagine dois sistemas equivalentes de E/S que são feitos de diferentes materiais físicos. Um sistema é feito de silício e recebe sinais elétricos como entradas e saídas, o outro sistema é feito de latas de estanho e cordas e recebe bolas de gude como entradas e saídas. Suponha que os dois sistemas sejam computacionalmente equivalentes de E/S. Em que poderia consistir sua equivalência de E/S? As respectivas entradas e saídas dos dois sistemas são diferentes, e podem ser tão diferentes que não possuem nenhuma propriedade funcional em comum. A única resposta parece ser que suas respectivas entradas e saídas representam a mesma coisa.

Conclusão "A única coisa que duas entradas e saídas fisicamente diversas têm em comum é que elas representam a mesma coisa".

> O problema é que não parece haver caracterizações funcionais não-semânticas suficientemente abrangentes para capturar todos os fatos relevantes sobre equivalência computacional.

Buscarei oferecer uma contribuição nesse sentido no próximo capítulo.

1.3.2.3 O argumento das distinções básicas

> Qualquer noção plausível de computação precisa fazer certas distinções básicas. A falha em fazer essas distinções marca uma falha em fornecer uma explicação adequada do cálculo. Uma dessas distinções básicas é entre portas E e portas OU, os blocos de construção de muitos computadores. Uma explicação da computação que não consegue capturar essa distinção não pode ser

Tabela 1.3: Uma implementação de uma porta E ou de uma porta OU?

in_1	in_2	out
0 V	0 V	0 V
0 V	5 V	0 V
5 V	0 V	0 V
5 V	5 V	5 V

adequada ou completa como uma explicação da computação.

Considere as duas tabelas de verdade 1.2. Suponha um sistema que retorne 5 volts (V) se ambas as entradas são $5V$, e retorna $0V$ caso contrário 1.3: este sistema implementa a porta lógica E ou a porta lógica OU?

Tabela 1.2: Tabelas de verdade dos operadores E e OU.

a	b	a E b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Sprevak nos dirá que "A diferença entre uma implementação de uma porta E e uma porta OU é uma diferença no conteúdo representacional". Contra essa conclusão, o terceiro capítulo desta monografia irá oferecer uma segunda alternativa, livre de pressupostos representacionalistas, baseada no recente artigo de Juliet Floyd. Tal alternativa consiste em pensar o uso de implementações como a imaginada por Sprevak segundo a noção wittgensteiniana de *forma de vida*. Antes, porém, é preciso voltar alguns anos na filosofia de Wittgenstein: no próximo capítulo, a concepção da matemática encontrada no *Tractatus Logico-Philosophicus* será contrastada com o a concepção da computação de 'On computable numbers...'.

2 DO NÚMERO DE WITTGENSTEIN AO NÚMERO COMPUTÁVEL DE TURING

> Computadores avançam,
> Artistas pegam carona.
>
> Fred 04

A filosofia da aritmética do *Tractatus* foi um laboratório para Wittgenstein. Ali ele desenvolveu uma notação para o cálculo aritmético que, como colocou Marion (MARION, 1998), prefigurara, em alguns aspectos, o cálculo-λ de Church. Como veremos, o cálculo tractariano, através de manipulações simbólicas regradas por algumas simples fórmulas, habilitou o seu autor a "demonstrar" que $2 \times 2 = 4$ (seção 2.2.2). Irei sugerir que tais fórmulas são o equivalente lógico de uma expressão bem formada em uma linguagem de programação qualquer. Em outras palavras, talvez a principal consequência da leitura da filosofia da aritmética que será oferecida neste capítulo será que Wittgenstein, ao definir o conceito de número, forneceu elementos de uma genuína *linguagem de programação*, tal como Raul Weber com a sua tabela de instruções do Neander e Alan Turing com sua teoria das máquinas. Como veremos na próxima seção, quando irei opor o conceito fregeano de número ao conceito de número de Wittgenstein, a mais tarde denominada por Frascolla de *teoria geral da operação* (FRASCOLLA, 2006) é tão simples quanto eficiente na delimitação de seu objeto: do ponto de vista da teoria da computação, o conceito de *número cardinal* é perfeitamente capturado pelo simbolismo tractariano.

Como veremos, talvez a principal consequência dessa interpretação seja que a demonstração oferecida por Wittgenstein seria melhor qualificada como computação. A partir de 'On computable numbers...',' manipulações operadas por uma máquina de computação passaram demonstrar a *computabilidade* de uma expressão: do número de Wittgenstein ao número computável de Turing, o principal movimento conceitual, como pretenderei mostrar, consistiu da descoberta do papel lógico do *computador*, a saber, o de servir como agente de manipulação simbólica.

2.1 Duas respostas para a pergunta "o que é um número?"

A filosofia da matemática é a disciplina que busca, dentre outras coisas, definições precisas para o conceito de *número*. Preocupa-se, desde ao menos a tradição iniciada por

Gottlob Frege (FREGE, 1978), em afastar da caracterização lógica deste conceito todo e qualquer traço de subjetividade. Deve-se, como nos diz o autor d'*Os Fundamentos da Aritmética*, "separar precisamente o psicológico do lógico, o subjetivo do objetivo". Em particular,

> ... a aritmética não tem absolutamente nada a ver com sensações. Nem tampouco com imagens mentais formadas a partir dos vestígios deixados por impressões sensíveis anteriores. A instabilidade e indeterminação de todas estas configurações opõem-se firmemente à determinação e estabilidade dos objetos e conceitos matemáticos. (*Os Fundamentos da Aritmética* (FA), p. 201)

A resposta à pergunta "o que é um número?" deve partir, portanto, de uma perspectiva inteiramente a priori: em outras palavras, para respondê-la deve-se posicionar *extra*-matematicamente, e a única maneira segura de fazê-lo, pensava Frege, seria sob o resguardo da lógica pura.

2.1.1 Número como extensão de conceito

No §68 dos FA, Frege, após passar em análise uma série de concepções acerca da aritmética, oferece a seguinte definição para o conceito de número

> Defino pois: o número que convém ao conceito F é a extensão do conceito "equinumérico ao conceito F".[11]

Para ele, o conceito de número deve ser capaz de capturar não apenas agrupamentos intuitivos, como o jogo de pedrinhas de Mill, mas todo o domínio do pensável. A rigor, lembra-nos Frege, tudo pode ser contado: desde objetos empiricamente distintos até a quantidade de chifres na cabeça do cavalo de Napoleão. De fato,

> As *verdades aritméticas* governam o domínio do enumerável. Este é o mais inclusivo; pois não lhe pertence apenas o efetivamente real, não apenas o intuível, mas todo o *pensável*.[12]

A afirmação de identidade numérica, portanto, é apenas um caso particular de afirmações universalmente válidas como a seguinte

$$\textbf{identidade}(7+5, \quad 12) \quad \{V, F\} \quad \rightarrow \quad V \iff 7+5=12$$

A distinção kantiana entre juízos sintéticos e analíticos[13] por muito operou im-

[11] FA, pp 253-254.

[12] Os Fundamentos da Aritmética, p. 217. Itálicos meus.

[13] Juízos da forma: A, que é B, é B (analítico) / A, que é B, é C (sintético). Uma caracterização que devo às aulas de Paulo Faria e pode ser encontrada nos cursos de lógica de Kant (KANT, 2004).

portantes conceitualizações no interior da filosofia. Frege, em particular, encontrou na dicotomia um meio de expressar suas ideias acerca da aritmética, mas para isso teve de adaptá-la: a noção kantiana não era suficientemente ampla para dar conta de algumas espécies de juízos relacionais. Considere, por exemplo, os seguintes juízos

Simetria Se Ken Thompson trabalhou com Dennies Ritchie na construção do *UNIX*, então Dennis Ritchie trabalhou com Ken Thompson na construção do *UNIX*.

Transitividade Qualquer músico que se influenciou no aspecto W por algum músico que se influenciou por Jaco Pastorius no aspecto W, se influenciou por Jaco Pastorius no aspecto W.

Certamente, não diríamos que a relação profissional de "trabalhar com" contém Ken Thompson e Dennis Ritchie, ou que a relação intelectual de "ser influenciado por" contém os músicos influenciados por Jaco Pastorius. Contudo, tais juízos complexos são analíticos. Para Frege, isso exigia uma mudança de olhar para as categorias fundamentais do juízo: as distinções entre sintético e analítico, e de a priori e a posteriori deveriam concernir à *justificação* da emissão do juízo. O desenvolvimento de sua *Conceitografia* serviu de anteparo para seu projeto maior de fundamentar a matemática sobre verdades lógicas.

> Se não é possível ... conduzir a demonstração sem lançar mão de verdades que não são de natureza lógica geral, mas que remetem a um domínio científico particular, a proposição é sintética. Para que uma verdade seja a posteriori requer-se que sua demonstração não se possa manter sem apelo a questões de fato, isto é, a verdades indemonstráveis e sem generalidade, implicando enunciados acerca de objetos determinados. Se, pelo contrário, é possível conduzir a demonstração apenas a partir de leis gerais que não admitem nem exigem demonstração, a verdade é a priori. (FA, p. 207)

Tomada amplamente, a atividade aritmética consiste da transformação de símbolos que estão por relações entre números em outros símbolos que igualmente estão por relações entre números. De fato, tal atividade pode ser exercida ainda que o usuário da matemática desconheça em absoluto qualquer definição do conceito de número: lembremos como, em nossas primeiras aulas de matemática, as operações básicas eram executadas correta ou incorretamente a revelia de todo mistério que a imaginação juvenil imprimia ao processo de cálculo. Contudo, o que seria dessa mesma atividade não fossem as sensações (entendida no sentido amplo de *experiência empírica*)? Com *o que* somaríamos? Ou ainda, *em que* subtrairíamos? Immanuel Kant notou que a dependência existente entre *operação aritmética* e a dimensão temporal de nossa apreensão de objetos é incontornável, posto ser inconcebível o desenvolvimento de qualquer série de número

sem passagem de tempo, e por esse motivo desenvolveu sua filosofia da aritmética subsidiariamente a sua estética transcendental. Frege, visando afastar-se da tradição kantiana de conceber a aplicação da aritmética como fundamentada essencialmente na estrutura de nossas *intuições*, propõe transpor as premissas de fundamentação para um sistema de dedução lógica. Acontece que, na época da publicação d'*Os Fundamentos da Aritmética*, a *Conceitografia* já havia avançado a tese de que todo juízo podia ser expresso em termos de *função* e *argumento*: assim, a estabilidade dos objetos e conceitos matemáticos poderia então, pensava Frege, subsumir-se à estabilidade de seu sistema lógico. Bastaria que a escolha dos axiomas, a partir dos quais todas as demais verdades matemáticas seguir-se-iam dedutivamente, fosse suficientemente acurada[14].

2.1.2 Número como expoente de operação

Para o autor do *Tractatus*, estava clara qual era a principal vantagem de substituir o vocabulário fregeano (de funções e argumento) pelo vocabulário operacional. Como categoricamente afirmado no aforismo 5.251, "Uma função não pode ser seu próprio argumento, mas o resultado de uma operação pode muito bem vir a ser base dela própria". De fato, como veremos, na base do conceito de número do TLP, estava a noção de *operação*.

"E assim", diz-nos Wittgenstein, "chegamos ao conceito de número: defino"

$$x = \Omega^{0\prime}x \quad Def. \tag{2.1}$$

$$\Omega'\Omega^{v\prime}x = \Omega^{v+1\prime}x \quad Def. \tag{2.2}$$

Como já se observou, a principal noção mobilizada no TLP $(6.02x)$ para a caracterização do conceito de número é a de *indução* (ou recursão). Tal ferramenta matemática consiste em oferecer uma definição do objeto em termos de si próprio (ACZEL, 1977). Dadas as definições (2.1) e (2.2), a série ordenada dos números naturais é gerada através da aplicação sucessiva das regras sobre uma mesma base arbitrária, produzindo a seguinte sequência:

[14] A esse respeito é interessante o trabalho de Alberto Coffa (COFFA, 1993), que busca mostrar, dentre outras coisas, como a filosofia da matemática logicista construiu-se sobre a analogia entre símbolos matemáticos e símbolos proposicionais.

$$
\begin{array}{rcrcrl}
x = \Omega^{0'}x & \rightarrow & 0 & = & 0 & Def. \\
\Omega' x = \Omega^{0+1'}x & \rightarrow & 0+1 & = & 1 & Def. \\
\Omega'\Omega' x = \Omega^{0+1+1'}x & \rightarrow & 0+1+1 & = & 2 & Def. \\
\Omega'\Omega'\Omega' x = \Omega^{0+1+1+1'}x & \rightarrow & 0+1+1+1 & = & 3 & Def. \\
\Omega'\Omega'\Omega'\Omega' x = \Omega^{0+1+1+1+1'}x & \rightarrow & 0+1+1+1+1 & = & 4 & Def. \\
\Omega'\Omega'\Omega'\Omega'\Omega' x = \Omega^{0+1+1+1+1+1'}x & \rightarrow & 0+1+1+1+1+1 & = & 5 & Def. \\
 & & \vdots & & &
\end{array}
$$

Em sua reconstrução sistemática da aritmética do TLP (FRASCOLLA, 2006), Frascolla oferece-nos a seguinte interpretação para os símbolos utilizados por Wittgenstein:

> x Exibe a forma de uma expressão que ainda não foi gerada pela aplicação de uma operação lógica.
>
> Ω Operação variável; símbolo para o conceito de operação.
>
> $'$ A forma do resultado da aplicação de uma operação a uma base dada; resultado de uma operação em geral.

Um "número", seja lá o que isso for, é incorporado pelo sistema tractariano *como* expoente de uma operação. Trata-se de um meio de codificação, e não de esclarecimento acerca de uma propriedade. Um pouco mais abaixo, Wittgenstein dirá que "o conceito de número é o número variável" (6.022) em um movimento de total esvaziamento ontológico de sua teoria: *número cardinal* não é propriedade de alguma coisa (como um conjunto arbitrário), mas sim de *operações* sobre coisas. Número cardinal, na aritmética tractariana, portanto, é definido como a marca deixada pela aplicação de uma operação sobre um objeto (o *expoente* da operação): seja o estádio do Maracanã em dia de jogo do Flamengo o *objeto* e a contabilização de torcedor a *operação*, se ambos estão precisamente definidos, então o resultado da operação nos oferecerá a cardinalidade do conjunto objeto.

2.1.2.1 A forma geral do número inteiro

Três são as ocorrências de *forma geral* no TLP: em 6, Wittgenstein introduz a forma geral da proposição; em 6.01, a forma geral da operação; em 6.03, por fim, a forma geral do número inteiro. Porém, como já foi observado (RODYCH, 2018), todas essas formas subordinam-se ao que Wittgenstein denominou de *termo geral de uma série*

formal (af. 5.2522), a saber, $[a, x, O'x]$, para a qual o próprio filósofo oferece a seguinte interpretação:

> ... Essa expressão entre colchetes é uma variável. O primeiro termo te expressão é o início da série formal, o segundo é a forma de um termo qualquer x da série, e o terceiro é a forma do termo da série que se segue imediatamente a x. (TLP, af. 5.2522)

A introdução da forma geral do número inteiro em 6.03 não é apenas uma abreviação da definição indutiva introduzida pouco antes. Mais do que isso, para Wittgenstein, seguer-se-ia do fato de que o conceito de número pudesse ser caracterizado como uma *forma geral*, que a série de números inteiros é, ela própria, uma série formal, e, portanto, codificável como termo geral de uma série formal.

> A forma geral do número inteiro é: $[0, \xi, \xi + 1]$. (TLP, af. 6.03)

Gostaria de argumentar que o conceito de número do TLP oferece as bases para uma crítica à tese fregeana acima mencionada de que "a aritmética não tem absolutamente nada a ver com sensações". Mais do que isso, oferece os meios para a abertura uma terceira via, pois, ao contrário de Kant, para Wittgenstein, a *intuição necessária* é fornecida pelo *processo de calcular* (6.2323). De um lado, portanto, o vienense se afasta da concepção fregeana de que a aritmética deve fundamentar-se em um sistema axiomático exterior ao sujeito de cálculo e, portanto, independente de sensações; por outro, ao mesmo tempo que julga necessário o papel da *intuição* no cálculo, se afasta da concepção kantiana da aritmética como dependente da estrutura transcendental de nossas faculdades cognitivas. Para Wittgenstein, a intuição é um sub-fenômeno da linguagem: no TLP, isso significava dizer que a intuição deveria estar codificada na forma geral da operação (a "forma geral da operação" é também a "forma mais geral da passagem de uma proposição a outra" (af. 6.01)); o conceito de número do TLP, sendo uma definição *indutiva* de número, não contém toda a extensão do conjunto dos números cardinais: são, antes, regras a partir das quais pode-se, por exemplo, efetuar a multiplicação de 2×2. Assim como uma receita de pão não contém o pão, uma definição indutiva de número não contém o número, mas sim o *meio* de construí-lo. Tal característica sugere o caráter construtivo da definição de número do *Tractatus*, aproximando seu autor do *convencionalismo* de Poincaré na mesma medida em que o afasta do *logicismo* de Frege. Nessa concepção, a "linguagem" deve ser o meio de codificação do número, que então pode ser compreendido como estrutura pelo decodificador.

2.2 A filosofia da aritmética no TLP

> Na vida, a proposição da matemática nunca é aquilo de que precisamos, mas utilizamos a proposição matemática *apenas* para inferir, de proposições que não pertencem à matemática, outras que igualmente não pertencem à matemática.
>
> – **TLP**, *6.211*.

A minha hipótese de leitura será que, nesta passagem do TLP, Wittgenstein está em diálogo direto com a crítica de Frege à concepção kantiana das proposições aritméticas como sintéticas. Segundo o vienense, ao substituir a intuição de Kant por um sistema axiomático, Frege não resolve nenhum problema matemático genuíno: tais problemas são encontrados na vida, e é com relação à vida que o resultado deve estar correto, não com relação a um determinado sistema de dedução lógica. Para Wittgenstein, Frege respondia a uma pergunta que ele próprio havia criado, uma pergunta que só fazia sentido no interior do novo sistema de cálculo lógico que, se revolucionou a lógica formal, não deve por isso ser tomado como mais fundamental que a matemática. Frege tinha em vista responder a pergunta "como demonstrar que $2 + 2 = 4$?". Wittgenstein dá um passo para trás como quem diz "o problema matemático genuíno está em como calcular $2 + 2$!".

> À questão de saber se a solução dos problemas matemáticos requer a intuição, deve-se responder que é precisamente a linguagem que fornece, nesse caso, a intuição necessária. (TLP, 6.233)
>
> O processo de *calcular* proporciona justamente essa intuição. (TLP, 6.2331)

No TLP, a resposta para a pergunta sobre como calcular será: através de regras de manipulação simbólica. A intuição (que, segundo Wittgenstein, é *necessária* para o cálculo), como nos diz na passagem citada, é fornecida pela linguagem. Sendo a intuição necessária fornecida pela linguagem, é no entanto *proporcionada* pelo processo de calcular. Esta ideia sem dúvida teria exercido forte impressão sobre Turing, se ele tivesse lido o TLP [15]: da concepção tractariana segue-se que o cálculo enquanto processo é capaz de proporcionar a intuição necessária para a resolução de problemas matemáticos. Contudo, na concepção wittgensteiniana, a análise da computação ainda não está completa: como já se observou, o sujeito tractariano é transcendental[16]. Wittgenstein fala sobre o processo de calcular, mas não sobre o sujeito que calcula. Talvez isso se deva ao fato de

[15] De fato, talvez Turing o tenha lido (considerando que mais tarde, em 1939, ele se interessaria pelas aulas acerca dos fundamentos da matemática oferecidas por Wittgenstein em Cambridge), mas desconheço qualquer referência que pudesse corroborar esta possibilidade.

[16] Com relação a este ponto (e vários outros deste capítulo), devo um agradecimento especial a Anderson Nakano, que me chamou a atenção para a inexistência do sujeito empírico no TLP. Também a Marcos Silva e a Pedro Noguez pelas frutíferas discussões acerca do tema na pacata e inspiradora Kirchberg.

que, como vimos, os computadores só receberam a primeira análise matemática cerca de 15 anos depois da publicação do *Tractatus*: Alan Turing, em 1936, domesticou a miraculosa máquina de resolução de problemas matemáticos com uma interessante e fecunda teoria, como relatado no capítulo anterior. Se estivermos corretos, em 'On computable numbers...', Turing domestica, também, o sujeito transcendental tractariano.

É importante observar que, no TLP, a *matemática* é um recorte bastante específico da disciplina: Wittgenstein tem em vista uma análise restrita ao mecanismo de operação das *equações*, e ainda que ele próprio de fato sugira que sua análise pudesse ser expandida para toda a matemática, não estarei interessado nesse aspecto do formalismo proposto por Wittgenstein[17]. O foco nas equações *aritméticas*, antes, evidencia que a empreitada logicista de Frege é problemática *de largada*.

No bloco de aforismos $6.23x$, o autor delineia os principais traços sua crítica ao logicismo fregeano. No que se segue, oferecerei uma reconstrução de alguns dos aspectos centrais dessa crítica. Tendo em vista conferir plausibilidade para a tese de que Wittgenstein teria oferecido elementos para uma análise da computação ao tomar as *equações* como paradigma central da matemática, a leitura a ser delineada partirá da suposição de um sujeito ao qual caberá operar os símbolos matemáticos. Em outras palavras, acrescentar-se-á um elemento que não estava presente na análise de Wittgenstein, mas que só viria surgir, anos depois, com a publicação do artigo de Turing, em 1936, a saber: a pressuposição do sujeito de cálculo.

> **6.23** Se duas expressões são ligadas pelo sinal de igualdade, isso quer dizer que são mutuamente substituíveis. Que seja esse o caso, porém, é algo que se deve mostrar nessas próprias duas expressões.
> Caracteriza a forma lógica de duas expressões serem elas mutuamente substituíveis.

Suponha que se diga "$2 + 3 = 5$": então, se a equação estiver correta, os símbolos $2 + 3$ podem ser substituídos por 5. Não se trata de demonstrar que S é P, mas sim de verificar que os passos do cálculo estão corretos. Suponha uma criança resolvendo, no quadro da sala de aula, problemas elementares de cálculo aritmético. Suponha agora que um dos problemas consistisse de somar o número 9 ao número 14. A professora, que observava atentamente seu aluno, notou, com curiosidade mas sem muita surpresa, que

[17] Paralelamente a essa questão, corre na bibliografia a discussão acerca do comprometimento de Wittgenstein com a existência de um procedimento de decisão para toda a lógica. Como se sabe, deve-se a Church e Turing as primeiras demonstrações matematicamente precisas de que tal *procedimento* era impossível. Em Turing, tal impossibilidade demonstrou-se via prova de não-computabilidade – como se verá ao longo do desenvolvimento deste capítulo, computabilidade será, em Turing, *codificabilidade*. Para uma discussão mais detalhada acerca desse debate no contexto da filosofia da lógica do TLP, faço referência a recente dissertação de Rodrigo Ferreira (FERREIRA, 2017)

o aluno obteve, após um cálculo efetuado com certo esmero, o resultado 104. Ora, não precisamos de muito para perceber que o resultado está patentemente errado, mas o aluno, ainda que receoso, foi capaz de manter a sua resposta com certo grau de confiança.

"Que isso seja o caso ... é algo que se deve mostrar nessas próprias duas expressões" – como a professora bem observara, o que aconteceu foi que o aluno equivocara-se na aplicação do "método da casinha" na soma de números com diferentes casas decimais: acontece que o aluno colocara o número 9 sob a dezena do número 14, transformando 9 em 90, e desse modo obteve o resultado de $90 + 14 = 104$. A professora, que percebeu o equívoco, foi e teria sido capaz de percebê-lo independentemente do fato de ser ou não capaz de oferecer uma prova de que o resultado correto deveria ser, na verdade, 23: isso porque, desde o ponto de vista da correção, é a adequação do cálculo com respeito à sua aplicação que conta como critério. A não substituibilidade de $9 + 14$ por 104 não reside no fato de que se pode provar que um não é idêntico ao outro, mas sim em que a expressão dessa igualdade (isto é, o processo pelo qual o aluno chegou ao resultado) não mostra que $9 + 14$ pode ser substituído por 104, mas sim que $90 + 14$ pode ser substituído 104: em outras palavras, o aluno oferecera a resposta correta para a pergunta errada.

> **6.231** É uma propriedade da afirmação que se possa entendê-la como dupla negação.
> É uma propriedade de "1+1+1+1" que se possa entendê-la como "(1+1)+(1+1)".

Nesta passagem, Wittgenstein atesta para o fato de que o símbolo matemático deve ser completo: deve bastar ao aluno o reconhecimento dos símbolos 9, $+$ e 14 para realizar a operação correta. Nesse ponto, a comparação entre os operadores matemáticos e os operadores lógicos é particularmente iluminadora: assim como a equivalência de proposições é uma propriedade da aplicação dos operadores, na matemática a equivalência de equações, que atesta a sua intersubstituibilidade, é uma propriedade da operação com os termos operandos.

> **6.232** Frege diz que as duas expressões têm o mesmo significado, mas sentidos diferentes.
> Mas o essencial, no caso da equação, é que ela não é necessária para se mostrar que as duas expressões ligadas pelo sinal de igualdade têm o mesmo significado, já que isso se pode ver nessas próprias duas expressões.

As interpretações acerca do *mostrar* no TLP divergem em variados pontos, e aqui não seria o lugar mais adequado para reconstruir as disputas no entorno do uso preciso que Wittgenstein faz do conceito[18]. Gostaria apenas de enfatizar uma característica que parece comum aos três aforismos citados acima: as expressões matemáticas são apresen-

[18]Para uma interessante discussão acerca do caráter simbólico do conhecimento veiculado pelo *exibir* tractariano, ver (SECCO; NOGUEZ, 2017)

tadas aqui como possuindo certa "virtude epistêmica", responsável por exibir, no símbolo mesmo, a *forma lógica* da expressão. Tal virtude, evidentemente, só ganha sentido na suposição de que tais símbolos são expressões veiculadas por algum meio físico.

> **6.2321** E que as proposições da matemática possam ser demonstradas nada quer dizer senão que sua correção é algo a ser visto, sem que deva o que exprimem ser comparado com os fatos quanto à sua correção.

No que diz respeito ao processo de resolução de uma equação, a correção independe integralmente dos fatos: como no caso da criança no quadro, que calculava "corretamente" (uma vez que manipulava os símbolos segundo as regras de um método) a despeito dos fatos observados. A criança, calculando corretamente $90 + 14$, ofereceu a resposta errada para o problema de resolver $9 + 14$: e Wittgenstein nos dirá que a percepção do erro não depende do reconhecimento do fato de que $23 \neq 104$, mas sim da observação dos passos internos ao próprio procedimento de cálculo.

2.2.1 Equação como manipulação regrada de símbolos

> O método pelo qual a matemática chega às suas equações é o método de substituição.
> Pois as equações exprimem a substituibilidade de duas expressões por outras expressões de acordo com as equações. (6.24) (SANTOS, 1993)

Compare "O céu é azul" com "$7 + 5 = 12$". A relação de identidade que o resultado 12 de uma soma mantém com a operação "$+$" sobre os operandos 7 e 5 não é a mesma que o termo sujeito "céu" mantém com o predicado "azul": para Wittgenstein, o sentido matematicamente relevante em que "$7 + 5$" (Φ) é "12" (Θ) atesta tão somente que $\Phi \rightarrow \Theta$, isto é, que Φ e Θ estão em relação de intersubstituibilidade[19]. Para o vienense, o que o sinal de igualdade mostra (poderíamos dizer: *indica*, no sentido de funcionar como uma *regra*), no contexto do cálculo aritmético, é que um pode ser substituído pelo outro.

A concepção da identidade numérica como intersubstituibilidade simbólica pode ser compreendida como opondo-se àquilo que Coffa (COFFA, 1993) chama de *proposicionalismo* na filosofia da matemática: a saber, a tendência de agrupar símbolos não propriamnente linguísticos sob a categoria de *proposição*, como ao dizer que "$7 + 5 = 12$" é uma proposiçao que expressa a igualdade de um objeto com relação a seu predicado.

[19] Tal como nos lembra Floyd, Whitehead, no capítulo dedicado à *natureza do cálculo* de seu tratado sobre a *álgebra universal* (WHITEHEAD, 1898), conceitualiza o *signo* (*sign*) do cálculo matemático como *substitutivo*. O método de substituição, nesta obra pré-*Principia* do matemático britânico, é introduzido no bem delimitado contexto de um cálculo equacional. A tradição algebrista, então representada por Whitehead, como argumenta Floyd, é incorporada por Wittgenstein em 6.24.

Wittgenstein se afasta dessa leitura na medida em que retoma a tradição, que remonta a Leibniz (LEGRIS, 2012), da álgebra como *calculus ratiocinator* e coloca a noção de *substituição simbólica* como central na atividade matemática: para Wittgenstein, o que uma equação *mostra* (para um sujeito transcendental, "o limite do mundo" que é o sujeito metafísico tractariano) é meramente que os símbolos são mutuamente substituíveis. Nesse sentido, é puramente formal o motivo pelo qual, dada a correção da igualdade matemática exemplificada acima, $12 + 3 = 15$ é matematicamente o mesmo que $(7 + 5) + 3 = 15$, por exemplo.

2.2.2 Demonstrando a proposição 2 X 2 = 4

6.241 Formula-se assim a demonstração da proposição $2 \times 2 = 4$:

$$(\Omega^{\nu})^{\mu}\text{'}x \quad = \quad \Omega^{\nu\times\mu}\text{'}x \quad Def., \tag{2.3}$$

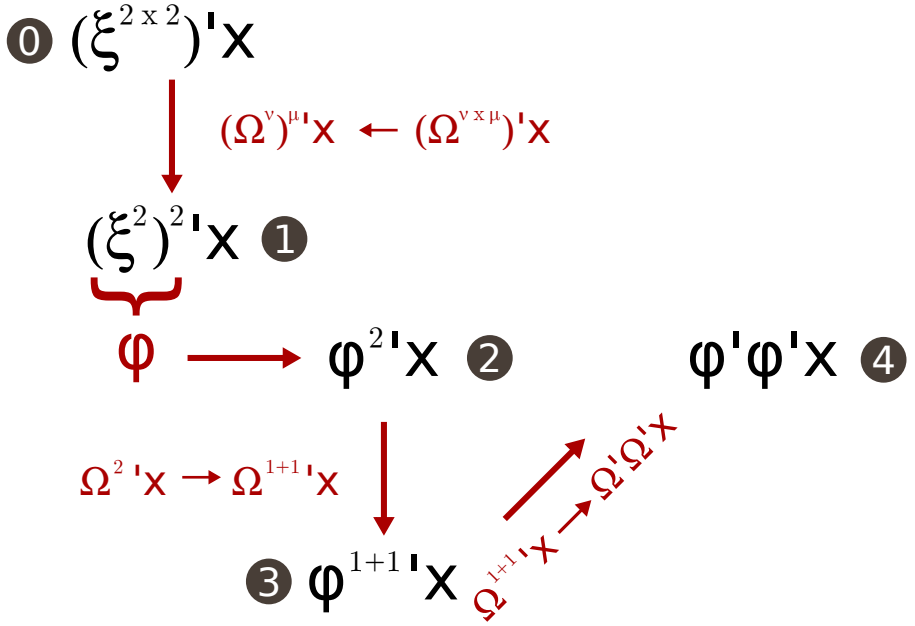
$$\begin{aligned}(\Omega^{2\times 2})\text{'}x \;&=\; (\Omega^{2})^{2}\text{'}x \;=\; (\Omega^{2})^{1+1}\text{'}x\\ &= \Omega^{2}\text{'}\Omega^{2}\text{'}x \;=\; \Omega^{1+1}\text{'}\Omega^{1+1}\text{'}x\\ (\Omega\text{'}\Omega)\text{'}(\Omega\text{'}\Omega)\text{'}x \;&=\; \Omega\text{'}\Omega\text{'}\Omega\text{'}\Omega\text{'}x \;=\; \Omega^{1+1+1+1}\text{'}x \;=\; \Omega^{4}\text{'}x.\end{aligned}$$

Para compreender esta demonstração, é preciso ter em mente não apenas o conceito de número introduzido por Wittgenstein ($[x, \xi, \xi + 1]$) e a definição 2.8 acima, mas também o método de substituição apresentado no aforismo imediatamente anterior (6.24). As figuras 2.1 e 2.2 esquematizam a demonstração em um diagrama de estados, cujos detalhes são oferecidos logo em seguida.

O primeiro ponto a ser mencionado é que, no diagrama de estados, é introduzida a variável auxiliar φ para substituir ξ^2. Isso porque, como observou Frascolla, Wittgenstein não oferece uma regra direta para transição de $(\Omega^n)\text{'}m$ para $(\Omega^n)^{1+1+\ldots+1_m}$', mas sim de Ω^n' para $\Omega^{1+1+\ldots+1_n}$'. O segundo ponto é que, do estado 7 para o estado 8, a transição não possui uma regra explicitada: nesse ponto, Frascolla nota a necessidade da introdução de uma regra explícita para a remoção dos parênteses; contudo, convém recordar o aforismo 6.231, quando Wittgenstein nos diz ser uma propriedade da afirmação de "$1 + 1 + 1 + 1$" que se possa entendê-la como "$(1 + 1) + (1 + 1)$". Optei por seguir a recomendação de Wittgenstein e deixar o reconhecimento da transição a cargo do *manipulador de símbolos*: o que nos leva ao nosso próximo ponto.

O terceiro e talvez mais importante ponto a ser observado é que a leitura que pro-

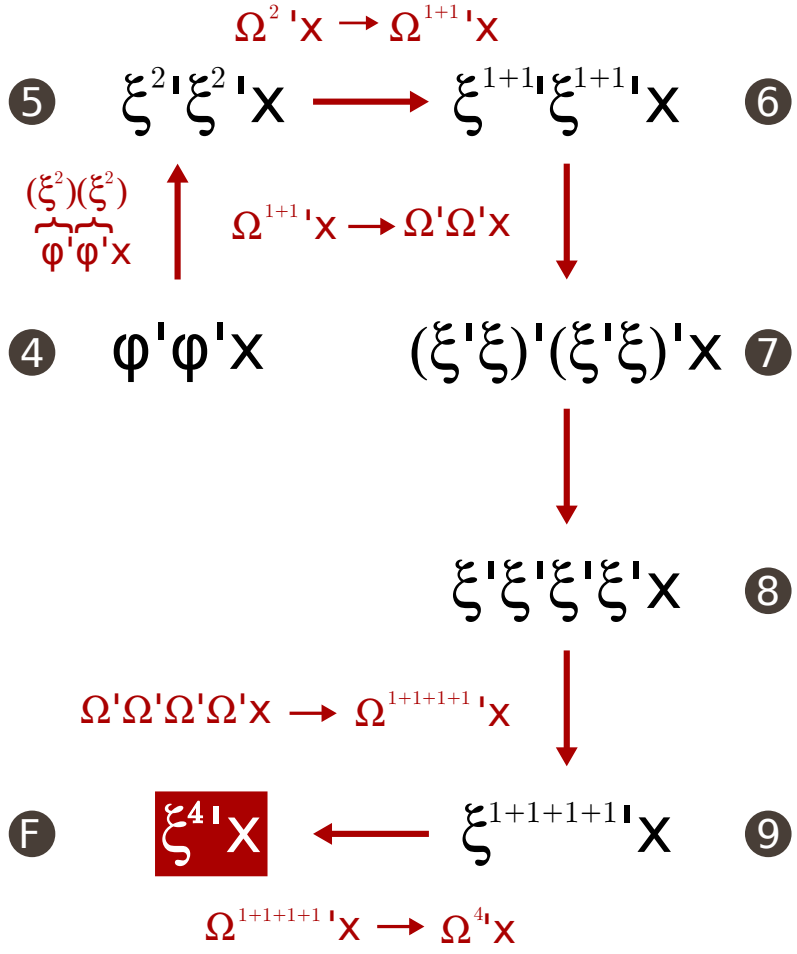
Figura 2.1: TLP 6241.



ponho para (6.241) acrescenta o *manipulador de símbolos* como responsável pela transição entre cada estado. Tal manipulador não estava explicitado na análise de Wittgenstein: argumento que ele, não obstante, completaria a análise de Wittgenstein na medida em que esclareceria o papel lógico do próprio método de substituição, atribuído (no *Tractatus*), explicitamente, à matemática. Por fim, o quarto ponto para o qual gostaria de chamar a atenção é que, em cada transição (com exceção da primeira, que decorre da aplicação da definição 2.8 formulada explicitamente no aforismo 6.241), optei por indicar o n-ésimo passo indutivo da aplicação da regra (isto é, a sua n-ésima iteração). Creio que, dessa forma, fica mais fácil de se reconhecer a substituição simbólica sendo operada na transição de um estado para o outro.

Como pode-se observar no diagrama, seguir as regras de substituição nos conduz ao símbolo ξ^{4}'x que, como vimos, nada mais é do que a quarta iteração de $[x, \xi, \xi + 1]$ – a quarta aplicação sucessiva de uma operação qualquer. Já que *número* é o expoente de uma operação, então ξ^{4}'$x \rightarrow 4$. Contudo, em que sentido poderíamos dizer que estaríamos "demonstrando" a proposição $2 \times 2 = 4$ simplesmente pela aplicação de regras pré-estabelecidas? Certamente, não no sentido forte, universal, visado por Frege: uma demonstração genuína deveria envolver, no mínimo, um conjunto consistente de afirmações verdadeiras. Wittgenstein, no entanto, recusa terminantemente que expressões matemáticas sejam afirmações de fatos acerca do mundo, "estado de coisas", e enfatiza que equações são pseudo-proposições: embora pareçam estar predicando, estão apenas *indicando* uma regra. A *equação* $2 \times 2 = 4$, portanto, não afirma nada além de que um símbolo pode

Figura 2.2: TLP 6241.



ser substituído por outro, dado um determinado conjunto de regras; e demonstrá-lo exige tão somente que apliquemos as regras, tal como realizado por Wittgenstein em 6.241 e no diagrama das figuras 2.1 e 2.2. Assim, o aforismo no qual culmina a concepção da aritmética do *Tractatus* oferece-nos uma demonstração em sentido *sui generis* da "proposição" $2 \times 2 = 4$. Se a leitura avançada nesta monografia estiver correta, tal sentido é o mesmo incorporado pela análise da computação formulada por Turing anos mais tarde.

Até este ponto, estivemos interessados em oferecer plausibilidade para a tese de que Wittgenstein, ao analisar as equações matemáticas, ofereceu uma análise da computação. O restante deste capítulo buscará estabelecer que tal análise teria sido completada por Turing em 1936, em sua redução da lógica da ação das máquinas a regras de manipulação simbólica. Como será melhor desenvolvido nas próximas seções, esta monografia defenderá que este "mostrar-se em si mesmo", no caso do processo operativo de resolução de equações matemáticas, depende de que *isto que se mostra* (a operação Ω simbolicamente formulada) esteja *naquilo que vê* (a entidade U que opera a partir do símbolo) em um sentido bastante particular, a saber, o de ser dado a U o *método de decodificação* de

Ω. Em outras palavras, Wittgenstein viu o Ω, mas não viu o U, que ganhou vida pela primeira vez com a teoria das máquinas Turing. Com base nessa concepção, será oferecida uma crítica ao segundo argumento de Sprevak. Antes, porém, é preciso retomar a teoria das máquinas de Turing, situando a tese **S** (seção 1.2.4) no contexto do artigo de 1936. Para tanto, na próxima seção, o conceito de número computável será introduzido e a sua relação com as equações tracatarianas, explicitada.

2.3 A lógica dos computadores

Por vezes, diante de algo que nos intriga, perguntamos "o que é X?" interessados nem tanto pela sua realidade quanto pelo mistério acerca da natureza incompreendida de um objeto que, a despeito de nossa perplexidade, não nos é inteiramente estranho e cuja utilidade inclusive se nos mostra nitidamente em nossas práticas cotidianas. A *teoria da computação*, tomada distintamente de outras áreas maiores como a matemática e a engenharia, é uma disciplina bastante recente. Creio que, ao menos como remédio temporário, possa resultar interessante buscar uma resposta para a pergunta "o que é um computador?" através de uma empreitada de cunho *filosófico*. Talvez por isso a disciplina de filosofia da computação recentemente tem encontrado lugar nos debates acadêmicos. Esta seção visará reconstruir a contribuição de Alan Turing (TURING, 1937) ao que poderíamos chamar de lógica dos computadores, a saber, à determinação dos critérios suficientes e necessários para a caracterização do objeto computador.

As pesquisas em lógica na primeira metade do século XX, principalmente após a publicação dos resultados de Gödel[20], resultaram em um projeto de definição matematicamente precisa para o conceito intuitivo de *procedimento efetivamente calculável*: de um lado, Alonzo Church et al. desenvolveram o cálculo-λ e, por meio dele, um modelo de computação para o cálculo algébrico. Turing, que publicou seu artigo algumas semanas depois de Church, buscou sua definição para o conceito de procedimento efetivamente calculável partindo de um problema de formulação mais cotidiana, a saber: quais seriam os critérios mínimos que um *computador* (compreendido como um ser humano empenhado na atividade de *resolução* de uma equação) deveria cumprir para ser capaz, ele próprio, de computar qualquer algoritmo que lhe seja entregue?

Como exposto no capítulo anterior, a teoria das máquinas de Turing modela tais

[20]Por meio de desenvolvimentos particulares que não caberiam ser aqui reconstruídos, mas a respeito dos quais faço referência ao artigo de Gandy (GANDY, 1988).

computadores como *máquinas de computação*. De acordo com a teoria, tais máquinas caracterizam-se (antes de tudo) por serem espécies de máquinas automáticas: máquinas cujo movimento é inteiramente determinado, a cada "estágio", pela sua própria tabela de instruções. Como veremos, o principal critério que um computador deve cumprir para ser capaz de executar um algoritmo do modo que Turing requer para suas máquinas automáticas é que as instruções que determinam o movimento sejam um sub-conjunto das instruções da própria máquina.

2.3.1 O que é um computador?

Uma maneira de responder a esta pergunta que, pelo menos à primeira vista, parece muito mais técnica do que propriamente filosófica, é apelando para a dicotomia *software/hardware*: um computador é esta espécie de pacto funcional existente entre os componentes físicos dos dispositivos computacionais (como os monitores LCD, discos rígidos, memórias *ram*, *joysticks*, etc.) e os seus componentes abstratos (isto é, os programas responsáveis tanto por coordenar os componentes físicos quanto por integrá-los).

Em um certo sentido, a presente monografia tem os *softwares* como objeto de estudo: na medida em que compreende-se por *software* todo e qualquer *código de máquina*, máquinas de Turing capturam apenas esta dimensão específica dos computadores. Nesse recorte específico, poderíamos introduzir o conceito de *máquina de Turing* da seguinte maneira: segundo o modelo proposto pelo matemático, todo *código de máquina* é, essencialmente, uma quintupla composta por

Alfabeto Conjunto finito qualquer de símbolos.

Cabeça Dispositivo de leitura e escrita.

Fita Sequência infinitamente longa de células nas quais a cabeça do computador pode escrever e mover-se para esquerda ou para a direita.

Estados Guardam a posição da máquina, por assim dizer, na lógica de sua estrutura interna. Estados "especiais": *inicial* e *final*.

Intruções Equacionam operação com símbolo lido pela cabeça na fita.

Uma máquina de Turing típica codificaria em estados a escrita do computador em função da operação primitiva de leitura. De fato, a estrutura de dados pressuposta pelo

modelo de Turing é consideravelmente simples se comparada com as variadas estruturas utilizadas hoje em dia. Em termos atuais, diríamos que a fita funciona como uma lista duplamente encadeada: ao computador de Turing caberia, então, percorrer linearmente a sequência de símbolos particularmente codificada na sua estrutura de dados, operando na medida de seus estados internos. Segundo este modelo, a mesma fita que serve para a *entrada* do computador, também serve para a sua *saída*.

Contudo, está longe de ser clara a fronteira pressuposta pela dicotomia software/hardware e, no que diz respeito especificamente ao formalismo proposto por Turing, aplicá-la é incorrer em ananacronismo: como já foi dito, o objeto de análise de Turing eram seres humanos, e não produtos de uma bem estabelecida indústria digital. No que diz respeito a esses computadores, uma máquina de Turing não é meramente um software, mas incopora também a dimensão do hardware: nesse sentido mais rigoroso, contudo, o *computador* não pode ser caracterizado como máquina de Turing sem qualquer qualificação adicional. Uma calculadora capaz de efetuar apenas as quatro operações básicas não tem o mesmo poder computacional que, por exemplo, o editor de texto *Emacs*, ou o interpretador de uma linguagem de programação qualquer. Tendo isso em vista, o passo seguinte da exposição consistirá em distinguir, de toda a infinidade de máquinas de Turing possíveis, a classe particular que, segundo a análise realizada pelo matemático britânico, é própria dos *computadores*: a saber, a categoria das *máquinas universais*.

2.3.2 Dos números computáveis às máquinas universais

> Os números "computáveis" podem ser descritos, de modo breve, como os números reais cujas expressões como um decimal são calculáveis por meios finitos. (p. 230)[21] (TURING, 1937)

O artigo no qual Turing introduz a sua concepção de procedimento efetivamente calculável é uma obra digna de nota em diversos sentidos. Gödel (como nos conta Gandy), certa vez, afirmou que a análise de Turing foi a primeira a oferecer uma definição absoluta (isto é, não dependente de um formalismo particular) para o conceito de *procedimento efetivamente calculável*. Se isso está correto ou não, o fato é que a clareza argumentativa do artigo de Turing facilita a leitura e coloca-nos em contato com a computação a um nível de abstração tal que a compreensão do procedimento torna-se independente da bagagem conceitual do leitor (diferente do que seria, por exemplo, a apresentação do conceito

[21] Em inglês, no original: "The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means."

segundo o formalismo do cálculo-λ).

A citação acima não apenas inicia o texto, mas sintetiza, em poucas palavras, três dos principais elementos da análise *computacional*: expressão, calculabilidade e finitude de meio.

Expressão Do ponto de vista da teoria da computação, números importam na medida em que são representáveis no interior de um modelo de cálculo. Contudo, a ideia da representação não se limita à representação numérica, e a possibilidade de conversão de um esquema representacional para outro perpassa, de fato, a inteireza dos processos computacionais. Contudo, o ato de representar, compreendido como o de *exprimir em termos de*, importa apenas durante a *programação*: é porque o programador sabe que, por exemplo, o número foi definido como "expoente de uma operação" que ele pode programar o algoritmo de soma em termos de expoentes de uma operação. Expressão é, em termos gerais, o *modo de apresentação* do dado.

Calculabilidade A análise de Turing é notável por atribuir um papel central aos computadores: o cálculo é a atividade realizada pelo *computador* durante o processo de *computação*.

Finitude de meio Algoritmos são, estritamente falando, conjuntos finitos de regras. Pode-se compreender intuitivamente a condição da finitude: considere o que seria uma receita de bolo cujo modo de preparo consistisse de infinitas instruções. Mais do que irrealizável, um tal bolo é incodificável: qualquer um que se impusesse a ingrata tarefa de escrever semelhante receita, por definição, jamais a concluiria.

Um número computável é, portanto, todo número cujo método de obtenção pode ser codificado. O número π ilustra bem o conceito: ainda que seja um número irracional, existe uma grande diversidade de *métodos de obtenção* codificáveis em máquina de Turing e, portanto, calculáveis. Como demonstrá-lo? Codificando. Certamente, uma codificação precisa de um algoritmo para o cálculo de π como máquina de Turing demandaria demasiados estados. Em vista de reduzir o emprego de esforços desnecessários, é frequentemente útil, em ciência da computação, escrever algoritmos como um *pseudo-código*. De certa forma isto é uma aposta do programador na computabilidade do algoritmo informalmente formulado, mas a prática mostra que a intuição bem treinada não costuma falhar para casos mais simples. Um algoritmo para a obtenção de π poderia ser

assim escrito em *pseudo-código*:

Algoritmo 1: Algoritmo para o cálculo de π escrito em *pseudo-código*.

Entrada: $\emptyset$

Saída: $\pi_{aprox} \in \mathbb{Q}$

início

  $k := 0;$

  $\pi_{aprox} := 0;$

  enquanto $k < \phi$ **faça**

    $\pi_{aprox} := \pi_{aprox} + (\frac{1}{16^k} * (\frac{4}{8*k+4} - \frac{2}{8*k+4} - \frac{1}{8*k+5} - \frac{1}{8*k+6}));$

  fim

fim

retorna π_{aprox}

De fato, o Algoritmo 1, acima, implementa um caso particular da famosa fórmula Bailey-Borwein-Plouffe (BAILEY; BORWEIN; PLOUFFE, 1997), cuja saída, de fato, tende ao número π com

$$\lim_{\phi \to \infty} f(x)$$

mas em um nível de abstração ainda bastante alto. Uma prova completa de computabilidade exigiria a formulação em termos mais estritos, e não caberia a sua realização na presente monografia. Gostaria apenas de sugerir aqui como poderiam ser os passos de uma tal prova: definir-se-ia, como máquinas de Turing, operadores lógicos (com base em que as condições poderiam ser reformuladas) e, igualmente como máquinas de Turing, cada uma das operações aritméticas envolvidas. Após tê-lo feito, integrar-se-ia cada uma das partes em uma única máquina de Turing: em outras palavras, ao se demonstrar que o algoritmo pode ser *codificado*, demonstra-se que ele pode ser *computado*. A situação é semelhante quando um motorista, confuso a respeito de qual lado deve dobrar na próxima esquina para chegar a seu destino, pergunta para um frentista: "Se eu seguir pela direita eu chegarei ao destino tal e tal?" e, ao obter a resposta positiva, conclui que se dobrar à esquerda ele *não* chegará ao seu destino. A inferência do motorista é indireta: o frentista não disse nada acerca do que aconteceria caso ele seguisse pela esquerda. No mínimo, o motorista pressupõe um certo grau de normalidade lógica em seu universo de ação. Semelhantemente, é por vias indiretas que a *computabilidade* de um algoritmo é demonstrada: um algoritmo é computável na medida em que pode ser computado por uma máquina *M*,

qualquer.

Ora, dado que todo algoritmo é, por definição, um conjunto finito de instruções, sua *codificação* deve poder ser obtida em tempo finito. Turing demonstra que é possível construir um único procedimento capaz de transformar cada máquina de Turing particular em um código (ao qual denominou de *descrição standard* do algoritmo) e, com isso, mostrou como uma única máquina pode ser capaz de decodificar cada uma dessas descrições standard, as quais passariam então a ser chamadas de *instruções* de uma máquina universal: conceitualmente, tal máquina seria o que hoje chamamos de *computador*. Como diz Floyd em referência a Davis (FLOYD, 2016),

> O que a máquina Universal implica, como Davis tem argumentado ... é que não existe tricotomia entre dado (entrada, experiência dada), software (regra, cálculo, inferência) e hardware. Ao invés disso, as fronteiras, aqui, também são fluidas.

De fato, o formalismo de Turing é utilizado no emprego da representação de algoritmos. Não quaisquer algoritmos *pensáveis* (como demandaria uma análise fregeana da computação), mas sim quaisquer algoritmos *calculáveis*[22]. A partir deste momento, dado que a análise de Turing da *calculabilidade* nos revela a irredutibilidade do computador, *calculável* será sinônimo de *computável*.

2.3.3 Compiladores como máquinas de Turing

Buscarei oferecer, nesta seção, uma explicação do mecanismo de compilação em termos de máquinas de Turing. A partir desta caracterização, como veremos, poderemos construir uma resposta livre de pressupostos semânticos ao problema da equivalência de entrada e saída. O *compilador*, como ficará evidente, opera livre de qualquer representação na *transformação* das instruções de uma determinada *linguagem* em outra (como, por exemplo, da linguagem $C++$ para código da máquina).

Para tanto, diremos que a linguagem de uma máquina universal M_x qualquer consiste de seu *conjunto de instrução*. Sejam $ISA1$ e $ISA2$ dois conjuntos de instrução quaisquer.

1. Se $ISA1 = ISA2$, então para cada $(x_n, \Omega_n)_1$ corresponde o mesmo $(x_n, \Omega_n)_2$.

2. Senão, pode-se construir um *compilador* para agir como *transformador sintático* na *tradução* de $ISA1$ para $ISA2$: o compilador é uma máquina de Turing que trans-

[22] Mais acerca deste tema será dito na última seção deste capítulo, quando a Tese de Church-Turing será brevemente discutida.

forma cada instrução de uma linguagem na instrução *operacionalmente equivalente* de uma outra linguagem qualquer, isto é, na instrução que realiza a mesma operação que realizaria na máquina para a qual a instrução transforma.

Suponha duas máquinas universais, $UTM1$ e $UTM2$, que operam segundo as linguagens $ISA1$ e $ISA2$, respectivamente. Digamos agora que o código para a instrução *soma* em $UTM1$ é ABC e o código para a instrução *subtração* nesta mesma máquina é DFG. Suponha, por fim, que os respectivos códigos de soma e subtração em $UTM2$ são RST e XYZ. O *compilador* da linguagem $ISA1$ para a linguagem $ISA2$ é meramente um programa de computador que opera as seguintes manipulações simbólicas

$$\begin{aligned} ABC &\rightarrow RST \\ DFG &\rightarrow XYZ \end{aligned} \tag{2.4}$$

2.3.4 A forma geral dos programas de computador

Segue-se da análise de Turing que todo conjunto de instrução ou bem é programa de computador, ou bem é programa de programa de computador, ou bem é programa de programa de programa de computador, e assim sucessivamente: em outras palavras, segue-se dessa análise que a forma geral dos programas de computador é recursiva (ver Figura 2.4). Supondo, na base C, um computador qualquer (uma máquina de computação), define-se um conjunto de instrução qualquer como o n-ésimo programa dado pela execução sucessiva de programas desse mesmo computador.

Assim, sendo P_0 definido como o conjunto total de instruções de um computador C qualquer, define-se o programa P_n do seguinte modo

$$P_0 \;=\; C \tag{2.5}$$

$$P_n \;=\; P_n(P_{n-1}) \tag{2.6}$$

Programa^1 $P_1(C)$

Programa^2 $P_2(P_1(C))$

Programa^3 $P_3(P_2(P_1(C)))$

Programa^4 $P_4(P_3(P_2(P_1(C))))$

$\vdots$

Programa^n $P_n(P_{n-1}(P_{n-2}(P_{n-...}(P_{(n-n)+1}(C)))))$

Figura 2.3: A semântica *right-first* do quinto programa da lista, $P5$, em notação de árvore.

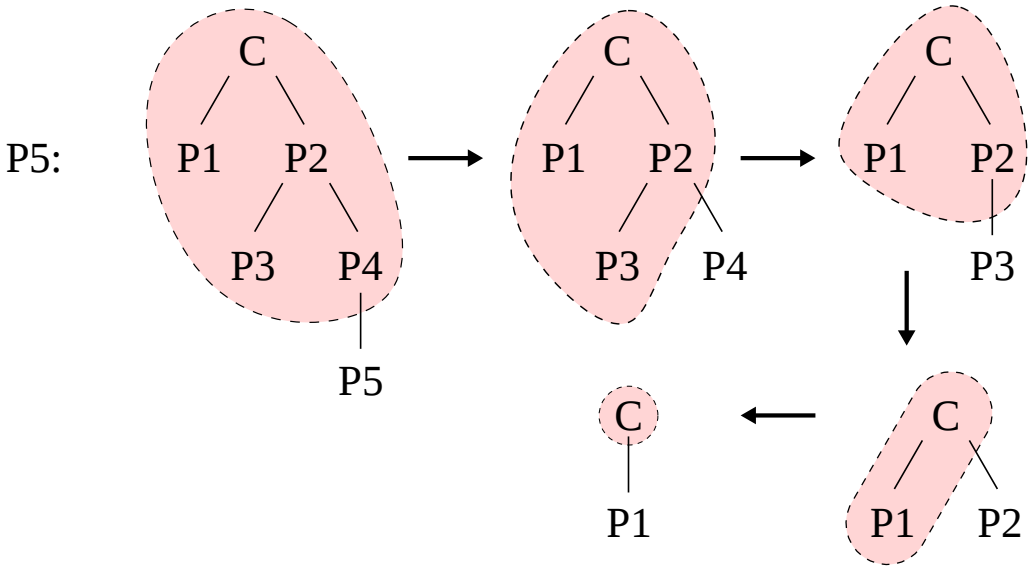
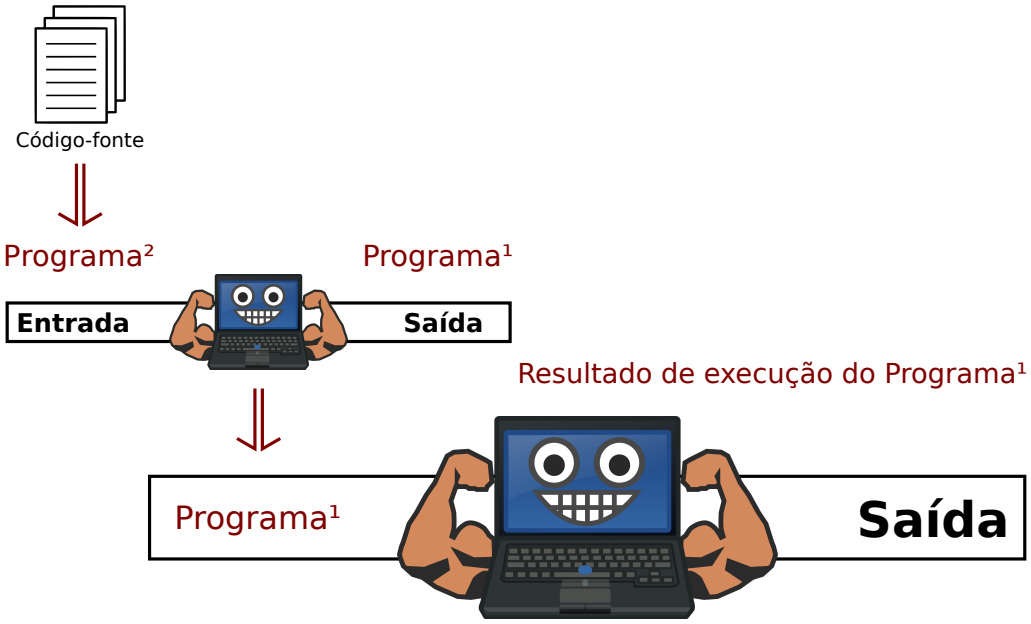


Figura 2.4: Forma geral de um programa de computador.



Fonte: clipart "computador pronto para o trabalho" <https://openclipart.org/detail/288754/computer-ready-for-work>, acessado em 23 de novembro de 2018.

Suponha, por exemplo, a tabela de instrução do Neander, tal como apresentada no capítulo anterior. Chamaremos nosso computador de N e iremos supor como primitiva

a regra ilustrada na computação de $P5$ (Figura 2.3). Desse modo, o algoritmo de soma poderia ser reescrito tal como apresentado na Figura 2.5 e na Figura 2.6.

Figura 2.5: Uma semântica para o algoritmo de soma do Neander.

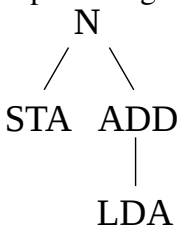
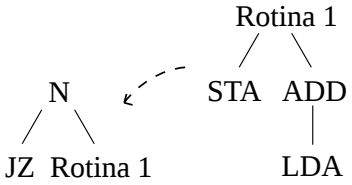


Figura 2.6: Cada nodo da árvore (LDA, ADD, STA) é, ele próprio, símbolo para alguma sub-rotina do computador. O acréscimo de JZ (*jump if zero*) sugere a flexibilidade permitida pelo uso de sub-rotinas.



No contexto da teoria das máquinas de Turing, a noção de sub-rotina está contida na denominada *tabela esqueleto*: tabelas de instrução abreviadas. Em linguagens de programação modernas (como o C, Java, $C++$), sub-rotinas tornaram-se centrais em vista da flexibilidade que oferecem para o programador. Se esta análise estiver correta, tais sub-rotinas são abstrações do computador real: no caso da *Rotina 1* (Figura 2.6), a abstração é feita com um rótulo arbitrário.

Se estivermos corretos, portanto, a definição recursiva acima produz uma série de programas de computador. Neste ponto, seria conveniente perguntar: a cada novo nível n de P, qual a relação entre P_n e P_{n-1}? Se levamos a sério a sugestão de Wittgenstein de que o método da substituição simbólica é o método *da matemática*, então a resposta seria: P_nRP_{n-1} tal que R é a relação intersubstituibilidade simbólica. Desse modo, poderíamos codificar a *série formal* da seguinte maneira

$$[C, P_n, P_{n+1}(P_n)] \tag{2.7}$$

Tal *forma geral*, evidentemente, aproveita-se do contexto tractariano. De fato, conjuga elementos da filosofia da aritmética do *Tractatus* com os aspectos apresentados da teoria das máquinas de Turing. Poderíamos dizer que tal forma geral é produto de um terceiro autor, que não é nem Turing (que não pensou em termos de formas gerais) e nem

Wittgenstein (que não pensou acerca de programas de computador), mas sim um autor que poderíamos chamar de Turing-Wittgenstein: tal autor aproveitou do TLP a ideia de que um número pode ser descrito segundo uma forma geral e a ideia de que a classe dos programas de computador é recursivamente enumerável do 'On computable numbers...' e introduziu a forma geral dos programas de computador como uma estrutura recursiva com base em um computador qualquer. Tal conceitualização permitirá, como pretendo mostrar na próxima seção, o delineamento final da nossa crítica ao representacionalismo na computação.

2.3.5 Uma (imaginada) resposta Turing-wittgensteiniana a Sprevak

Como antecipado na seção 2.2, nesta seção se buscaria explicar o *mostrar-se em si mesmo* dos símbolos matemáticos tractarianos como dependentes de um estar em posse do método de decodificação. Se estivemos certos até aqui, a única dependência essencial da computação é a existência de um computador. Quando Wittgenstein conceitualiza o número como expoente de uma operação, ele oferece um meio de codificação desses números: nessa medida, o conjunto de instruções do computador pode construir-se sobre essa codificação específica, assim como, nos dias de hoje, a codificação binária é utilizada nos mais variados níveis de abstração em que se pode considerar um computador digital. Dessa forma, a posse do método de decodificação é, em termos gerais, o que possibilita ao computador operar sobre as diversas estruturas de dados codificáveis em termos de seu conjunto de instruções. Nessa medida pode-se pensar aquilo que se mostra no símbolo matemático como aquilo que se codifica na linguagem de máquina. Nesse caso, como fica claro, expressões matemáticas passam a ser instruções, e o computador o agente de decodificação.

No contexto tractariano, o conceito de representação entra *via* o de proposição: no TLP, proposições representam estados de coisa no mundo na medida em que existe uma relação de isomorfismo entre os elementos simples da proposição (produtos de uma análise completa) com os elementos constitutivos do (possível) espaço lógico representado. Nesse sentido, valores de verdade existem como parasitários de proposições: os inevitáveis *pólos* da proposição em que consiste a tese da bipolaridade define-os como contingentes estados de uma configuração simbólica particular. "Chove agora." está escrito nesta monografia, e, como *sinal*, assim sempre estará enquanto exista alguma cópia deste texto: *qua* símbolo de uma proposição, carrega o fardo de dois pólos, variáveis em

relação às condições metereológicas, e característicos da capacidade semântica da expressão.

Nesses termos, fica claro por quê a teoria axiomática de Frege enfrenta problemas ao atribuir verdade e falsidade a proposições matemáticas. Equações, nesse sentido, não são proposições, pois nada representam. A utilidade de tais expressões, ou, melhor dizendo, o seu papel lógico, deve derivar de alguma outra característica que não a de representar verdadeiramente algum estado de coisas. Como vimos na leitura oferecida da concepção tractariana da matemática, expressões da forma $\Phi = \Sigma$ (como $7 + 5 = 12$), se *corretas*, demonstram a intersubstituibilidade de Φ por Σ (no exemplo, $7 + 5$ por 12). A noção de correção, na medida em que não depende da noção de verificação, é-nos bastante útil: uma expressão é correta se, e somente se, resultar unicamente da aplicação das regras do método. No aforismo 6.24 Wittgenstein torna explícito o método que tem em vista: equações, diz o filósofo, são regras de *manipulação simbólica*.

Como visto (seção 1.3.2), a tese **R** afirma que conteúdos representacionais são *essenciais* para a computação. Vimos também como tal afirmação compromete o representacionalismo com a visão segundo a qual a comunicação indireta (via linguagens de programação) envolveria a transmissão de alguma espécie de conteúdo semântico. Uma análise da computação com base, em primeiro lugar, na teoria das máquinas de Turing e, em segundo, a filosofia da aritmética do *Tractatus*, permitiu-nos conceitualizar programas de computador como o n-ésimo programa de um computador C qualquer. Tal análise sugere a intrínseca dependência entre expressões bem formadas de uma linguagem de programação qualquer e as equivalentes instruções do computador C, obtidas, segundo a teoria das máquinas de Turing, inteiramente através de manipulação de símbolos.

A demanda pela existência de conteúdos representacionais veiculados por processos computacionais não é nova. Na origem de tal demanda estão algumas correntes da filosofia da mente que buscaram explicar o funcionamento do fenômeno mental a partir de modelos de computação. Como já foi observado (HASELAGER; GROOT; RAPPARD, 2003), "a suposição de que cognição consiste em manipulação computacional de representações é uma suposição básica na ciência cognitiva" (p. 6). Contudo, se Turing oferece uma análise final do que seja computar, tal demanda deve ser vista como extrínseca à teoria da computação: estritamente falando, de acordo com a teoria das máquinas do matemático britânico, o fenômeno da computação se reduz a um processo de manipulação simbólica que é levado a cabo pelo computador que movimenta-se pelo meio físico de modo inteiramente determinado pelas suas configurações internas. Os computadores

de Turing são verdadeiros mestres na técnica do *pensamento cego* leibniziano[23]. Desse modo, no mínimo, a tese **R**, ao afirmar uma necessidade (a saber, que a computação *deve* envolver representação), afirma uma falsidade. Se dermos um passo além na caracterização desta crítica Turing-wittgensteiniana, poderíamos dizer que, na medida em que confunde a aplicação de conceitos na suposta formulação de um juízo, o representacionalismo na computação incorre em exemplar caso de contra-senso.

[23]Em (DAVIS, 2011), Davis apresenta Turing como um herdeiro direto de Leibniz. Em nossa reconstrução, buscamos sugerir que a filosofia da aritmética de Wittgenstein também poderia ser enquadrada na mesma linha, ainda que o trabalho de traçar precisamente tais relações ainda esteja por ser feito. No caminho de tal aproximação estão os importantes trabalhos (CASANAVE, 2012) e o já citado (SECCO; NOGUEZ, 2017).

3 A *LEBENSFORM* DA PROGRAMAÇÃO DE COMPUTADORES

> O senhor... Mire veja: o mais importante e bonito, do mundo, é isto: que as pessoas não estão sempre iguais, ainda não foram terminadas – mas que elas vão sempre mudando. Afinam ou desafinam. Verdade maior. É o que a vida me ensinou. Isso me alegra, montão.
>
> *Grande Sertão: Veredas*, João Guimarães Rosa

A atividade de programação de computadores existe ao menos desde que os seres humanos escreveram o primeiro algoritmo: nos primórdios, contudo, a única ferramenta existente para tal propósito derivava de formas linguísticas ainda rudimentares para a tarefa de lidar com as demandas da atividade. Apenas recentemente, a "linguagem perfeita" transfigurou-se de um sonho de Leibniz para notação na *Conceitografia* de Frege. A descoberta realizada por Kurt Gödel de que nenhuma perfeição poderia ser completa, longe de reduzir os feitos fregeanos, antes conduziram-nos pelo caminho do que Robin Gandy denominou de a confluência de ideias de 1936.

Na passagem citada de Guimarães Rosa, a diversidade da vida humana é avaliada desde a perspectiva de sua inerente incompletude: em sua fala, Riobaldo enfatiza a permanente mudança a que estamos todos sujeitos enquanto *pessoas*. Para que tenhamos dimensão do quão gratificante é o fato de que as coisas sejam assim, basta que comparemos com a hipótese de que as coisas não fossem dessa maneira: nesse caso, aquilo que verdadeiramente somos estaria dado de antemão, como um produto pronto de fábrica. Nesse caso, certamente, seríamos a imagem e semelhança dos computadores. Não seríamos, portanto, pessoas: os mecanismos pelos quais os computadores operam, como vimos, embora possam ser simulados por uma pessoa, não reduzem o universo da vida humana, mas sim o universo dos procedimentos efetivamente calculáveis. Convém rejeitar, portanto, toda característica genuinamente humana na caracterização das máquinas: máquinas não pensam, não amam, não sentem e, no sentido de Riobaldo, não mudam. Seres humanos, não obstante, mudamos: o caso paridgmático de mudança a ser considerado no desenvolvimento do último capítulo desta monografia será o da mudança na nossa

relação com os próprios computadores.

3.1 Um retorno aos problemas centrais

A partir da invenção do conceito de programa armazenado, uma das primeiras coisas aprendidas por um indivíduo que resolve iniciar seus estudos em ciência da computação é diferenciar entre arquivos de tipo *executável* de arquivos de *dado*. O primeiro, aprende-se, contém instruções codificadas de modo tal a instruir a máquina (na memória da qual está alojado) acerca de uma determinada tarefa; o segundo tipo inclui todos os arquivos cuja estrutura não é diretamente compreensível pela máquina, e por isso depende de outros programas para torná-los "significativos": o caso notável é o dos arquivos de mídia (filmes, músicas, livros, etc), mas também devemos incluir nessa lista toda espécie de código-fonte – seja ele escrito em *C++*, *Java*, *Shell Script*, *Python*, ou em qualquer outra linguagem de programação distinta da linguagem primitiva da máquina.

Pressupondo a teoria das máquinas de Turing, estamos em condições de responder a duas das questões centrais desta monografia, a saber:

O que é um computador? Computadores são máquinas de computação.

Como funcionam? Lendo, escrevendo, e movimentando-se célula por célula em uma fita potencialmente infinita.

Foi afirmado acima que a estrutura de um arquivo de dado não é diretamente "compreensível pela máquina" e isto *deve* ser esclarecido. Como pretendi mostrar, tal esclarecimento nos conduziu à resposta da questão sobre como as máquinas se comunicam (entre si e conosco): com base na teoria das máquinas de Turing, a partir da qual dividimos a comunicação de máquina em *direta* e *indireta*, descobrimos que em ambos os casos a comunicação estabelecida se dá via o mesmo princípio, afirmado pela **Tese S** e descrito a seguir.

Como computadores se comunicam? Manipulando símbolos.

Como vimos, os símbolos movimentados durante o processo de computação não necessitam de qualquer representação que lhes confira conteúdo semântico. Nesse sentido é que se diz que dispositivos computacionais são puramente mecânicos: toda ação é pré-determinada por uma configuração interna e relativa à lógica do computador. Uma

das consequências da análise de Turing, como pretendi argumentar, é que toda representação adventícia seja por conseguinte inteiramente extrínseca à computação. Desse modo, estamos em condições de distanciar-nos, enquanto seres dotados de linguagem, de tais objetos: a comunicação humana (isto é, a comunicação possível entre seres humanos), ao contrário da comunicação de máquina, como será brevemente sugerido na próxima seção, abrange uma tão variada quando complexa estrutura semântica.

3.2 Programação de computadores como forma de vida

A *natureza* humana, como nos mostra Arendt (ARENDT, 2013), seria melhor caracterizada como *condição*, posto que, em sua formulação, o conceito de *ser humano* deve abranger a inevitável *pluralidade* da *vida* humana, tão característica da nossa espécie e tão necessária para nossa plena realização como *pessoas*. Hannah Arendt, em *A Condição Humana*, enfatiza a dimensão do diálogo na relação do indivíduo com seu meio: afinal, por que outra via a complexa estrutura da vida prática dos seres humanos em comunidade poderia ser organizada? A capacidade de comunicar pensamentos, sem dúvida nenhuma, constitui um dos maiores tesouros legados pela evolução de nossa espécie. Naturalmente, tal processo de desenvolvimento da linguagem humana dependeu de diversas pressões ambientais, as quais conduziram os organismos de nossa linha evolutiva à criação e ao aprimoramento dos mecanismos de comunicação linguística.

Se estivemos corretos até aqui, a linguagem humana possui ao menos um mecanismo inexistente nas linguagens de máquina: a saber, o de veicular *representações* de estados de coisa. Contudo, poder-se-ia argumentar que existe uma diversidade de outros mecanismos da linguagem humana que não são capturados pela linguagem de máquina: o de expressar emoções, por exemplo. Um programa de computador poderia ser ferramenta extremamente útil para auxiliar na análise de um soneto de Vinícius de Moraes, mas, tanto quanto diz respeito à compreensão mesma do texto, tais programas não exercem nenhum papel: isso porque o sentido relevante de *compreensão*, no que diz respeito à leitura de uma poesia, está inevitavelmente atrelado à capacidade humana de fruir artisticamente a partir de uma determinada configuração simbólica.

Aprendemos a ler poesia, e isto, dir-nos-á Wittgenstein em sua obra de maturidade, é uma forma de vida. Contar uma história, outra forma de vida. Representar teatro, relatar um acontecimento, etc., são outros exemplos que Wittgenstein oferece, no vigésimo terceiro parágrafo das *Investigações Filosóficas*, do que seja uma *forma de vida*.

Dentre esses exemplos, um deles consiste em “comandar, e agir segundo comandos”. Se a comunicação (direta ou indireta) é o meio pelo qual seres humanos comandam computadores, penso que não seria demais sugerir, como uma primeira aproximação, que a forma de vida da programação de computadores seria característica da forma de vida de “comandar”, ao passo que a forma de vida dos próprios computadores seria a de “agir segundo comandos”.

Como formas de vida, tais atividades estão sujeitas a toda espécie de alteração: em ‘Chains of life: Turing, Lebensform, and the emergence of Wittgenstein’s later style’, Juliet Floyd sugere que isso está relacionado a “fluidez” dos procedimentos humanos.

> Wittgenstein se afastou, conceitualmente, da figuração axiomática vertical da hierarquia cima-pra-baixo e baixo-pra-cima de Russell (e Gödel): de um arquipélago desordenado de modelos de cálculo [*calculi*] bem-fundados porém convencionais, para uma ordem serial de jogos de linguagem e, por fim, para uma fluida configuração modular de procedimentos humanos, embutidos em nossos modos ordinários de falar, tal como estes evoluíram de nossa vida cotidiana. A análise de Turing da lógica fornece justamente uma tal visão. Nosso uso rotineiro de “aplicativos” hoje evidencia e instancia a perspectiva. Eles são *Lebensformen*.

No *Tractatus Logico-Philosophicus*, diz-nos Floyd, Wittgenstein ainda estava às voltas com uma noção do *simples* como algo absoluto: o que a análise final de uma proposição relevaria seria sempre o mesmo, independente de qualquer condição externa à própria análise. Tal visão, penso eu, sugere uma imagem da linguagem proposicional cuja estrutura se assemelha, em grande parte, ao que temos visto sobre linguagens de programação, com a diferença fundamental de que expressões bem formadas de uma linguagem de programação, ao contrário de uma proposição genuína, nada representam: nesse sentido (e esse foi o argumento do segundo capítulo desta monografia), toda expressão de regra é uma pseudo-proposição. Ao longo do desenvolvimento de sua obra, como argumenta Floyd no artigo supracitado, o vienense “afastou-se” da concepção hierarquizada da linguagem que remonta a Russell e Gödel. Para o Wittgenstein das *Investigações Filosóficas*, dialogar é praticar um (ou vários) jogo(s) de linguagem, e nenhum jogo é mais ou menos fundamental que outro: todos, antes, compõem e configuram o rico e complexo quadro da linguagem humana.

Floyd, nesse artigo, realiza um trabalho semelhante ao realizado na presente monografia. Contudo, ao contrário de nossa proposta, a autora está interessada em traçar relações de mútua influência entre Wittgenstein e Turing e, para tanto, busca no conceito de *forma de vida* elementos em favor dessa aproximação. Segundo ela, tal conceito surge na obra do vienense como produto da resposta madura do filósofo à pergunta sobre a *natureza do agir logicamente* após convencer-se da análise de Turing para o conceito de

"passo-a-passo" em um sistema formal. Tal análise corresponde ao que apresentamos, neste trabalho, sob o nome de *teoria das máquinas*.

Computadores agem logicamente. De fato, computadores agem *apenas* logicamente. Desse modo, uma resposta para a pergunta acerca do *lógico* deveria nos conduzir a uma compreensão, também, acerca da natureza da ação dos computadores. Como sugeri acima, talvez a chave esteja em pensar programas de computador como casos específicos de um jogo de linguagem particular, a saber, o de *comandar e agir segundo comandos*. O desenvolvimento de linguagens de programação, bem como o uso cotidiano de ferramentas digitais, desse modo, poderia moldar-se sobre novas bases. Penso que uma das principais consequências dessa análise seria pragmática: devemos construir programas de computador capazes de *participar* de nossos jogos – e não de jogá-los por nós[24].

Nesse sentido, o terceiro argumento de Sprevak, que sugere a necessidade da representação a partir do problema da ambiguidade entre diferentes implementações de uma mesma operação, perderia a sua força. A pergunta "Qual função o sistema da Tabela 1.3 implementa?" simplesmente não possui resposta, pelo menos até que o programador leia as especificações funcionais no manual do sistema. Conceitualizar programação como *forma de vida* é, antes de mais nada, tornar-se consciente da incompletude essencial de certos sistemas formais consistentes. Certas questões, como aquela pela qual Sprevak esperava sugerir a necessidade de se atrelar conteúdos representacionais a constantes lógicas (uma outra ideia que o Wittgenstein do *Tractatus* certamente rejeitaria, a julgar pelo seu *Grundgedanke* [af. 4.0312]), simplesmente não podem ser respondidas antes de que outras informações sejam fornecidas. Tais informações são justamente aquelas que só podem ser obtidas na fluidez do dia a dia e através do *uso* dos computadores.

3.3 Conclusão

Talvez a maior lição que podemos extrair do artigo de 1936 de Turing seja que *não existe computação sem computador*. Deixando de lado a aparente trivialidade da afirmação, isso quer dizer que a teoria da computação, em vista daquilo de que é teoria, depende de modelos matemáticos capazes de capturar o conceito de computador de forma precisa. Dentre as diversas espécies desses modelos, como argumentamos, as linguagens

[24] E se criamos programas de computador capazes (até certo ponto) de jogar alguns de nossos jogos, como uma Inteligência Artificial programada para jogar xadrez, que seja porque tais programas podem ser usados em favor de outros jogos, como o jogo de ensinar uma criança a arte enxadrista.

de programação são exemplos paradigmáticos.

Como busquei mostrar ao longo do desenvolvimento desta monografia, existe uma diferença fundamental que distingue o ser humano dos computadores. Para tanto, argumentei que, em 'On computable numbers...', Turing se compromete com um dos paradigmas centrais da computação dos dias de hoje, a saber, o paradigma da computação como manipulação regrada de símbolos. Como vimos, tal distinção torna-se notável quando comparamos seres humanos e máquinas em termos de suas respectivas capacidades linguísticas. Perto da complexidade do fenômeno da *fala* humana, como Gilberto Gil parece sugerir na canção que abre o presente trabalho, a linguagem de máquina é um mero brinquedo. Contudo, como nos sugeriu Fred 04 no segundo capítulo desta monografia, é preciso compreender o papel desses computadores em nosso dia a dia para que *nós*, seres humanos e verdadeiros *artistas*, saibamos pegar carona no avanço tecnológico. Guimarães Rosa, através da simples, porém profunda, sabedoria do sertanejo Riobaldo, chama a nossa atenção para um fato característico da nossa humanidade: a saber, o fato de que todos nós estamos sempre sujeitos a um processo de permanente mudança, pelo qual erramos, acertamos, afinamos e desafinamos: mas o mais importante, e bonito, de tudo isso, é que a mudança faz de nós aquilo que somos.

No primeiro capítulo desta monografia, através da reconstrução da teoria das máquinas de Turing, busquei mostrar como a tese da computação como manipulação regrada de símbolos é central no modelo proposto pelo matemático britânico. No segundo capítulo, a pergunta em aberto acerca da natureza da comunicação dos computadores foi examinada a partir do caso paradigmático da programação de computadores, com base em que aproximamos o conceito de número do *Tractatus Logico-Philosophicus* do conceito de número computável de Turing em 'On computable numbers...'. Tal aproximação conduziu-nos a uma caracaterização da comunicação computacional como inteiramente determinada por manipulações simbólicas. Por fim, no último capítulo deste trabalho, introduzimos a noção de programação como *forma de vida* e apresentamos alguns aspectos e impactos na prática do uso dos computadores.

Como um possível desdobramento da pesquisa realizada para esta monografia estaria um exame mais aprofundado das relações conceituais entre Leibniz e Wittgenstein, em particular no que diz respeito aos aspectos explorados neste texto do conceito tractariano de número inteiro. Tal exame, sem dúvida, estreitaria as relações entre o vienense e Turing, sendo este último herdeiro direto de Leibniz. Além disso, os desenvolvimentos posteriores da filosofia da matemática de Wittgenstein parecem apontar para um certo

construtivismo que, se incorporado a esta pesquisa, sem dúvida poderia revelar novos e interessantes aspectos dessa relação. Por último, mas não menos importante, estaria um aprofundamento do estudo dos debates travados entre Wittgenstein e Turing em 1939, ao longo do período em que o matemático frequentou a sala de aula do filósofo. Este trabalho, em particular, Juliet Floyd vem fazendo com muita competência nos últimos anos, alimentando o potencial de tal análise revelar importantes traços das posições de ambos pensadores.

REFERÊNCIAS

ACZEL, P. An introduction to inductive definitions. In: **Studies in Logic and the Foundations of Mathematics**. [S.l.]: Elsevier, 1977. v. 90, p. 739–782.

ARENDT, H. **The human condition**. [S.l.]: University of Chicago Press, 2013.

BAILEY, D.; BORWEIN, P.; PLOUFFE, S. On the rapid computation of various polylogarithmic constants. **Mathematics of Computation of the American Mathematical Society**, v. 66, n. 218, p. 903–913, 1997.

BOOLOS, G. S.; BURGESS, J. P.; JEFFREY, R. C. Computabilidade e lógica. **São Paulo: Editora Unesp**, 2012.

CASANAVE, A. L. **Symbolic Knowledge from Leibniz to Husserl**. [S.l.]: College Publications, 2012.

COFFA, J. A. **The semantic tradition from Kant to Carnap: To the Vienna station**. [S.l.]: Cambridge University Press, 1993.

COPELAND, B. J. **Alan Turing's Automatic Computing Engine : The Master Codebreaker's Struggle to build the Modern Computer: The Master Codebreaker's Struggle to build the Modern Computer**. [S.l.]: OUP Oxford, 2005. ISBN 978-0-19-152410-3.

DAVIS, M. The Myth of Hypercomputation. In: TEUSCHER, C. (Ed.). **Alan Turing: Life and Legacy of a Great Thinker**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 195–211. ISBN 978-3-662-05642-4. Disponível em: <https://doi.org/10.1007/978-3-662-05642-4_8>.

DAVIS, M. **The universal computer: The road from Leibniz to Turing**. [S.l.]: AK Peters/CRC Press, 2011.

DIVERIO, T. A.; MENEZES, P. B. **Teoria da Computação–UFRGS: Máquinas Universais e Computabilidade**. [S.l.]: Bookman Editora, 2009.

EGAN, F. Individualism, Computation, and Perceptual Content. **Mind**, v. 101, n. 403, p. 443–59, 1992.

FERREIRA, R. S. A lógica do Tractatus e o operador N: decidibilidade e capacidade expressiva. 2017.

FLOYD, J. Chains of Life: Turing, Lebensform, and the Emergence of Wittgenstein's Later Style. **Nordic Wittgenstein Review**, p. 7–89, dez. 2016. ISSN 2242-248X. Disponível em: <https://www.nordicwittgensteinreview.com/article/view/3435>.

FRASCOLLA, P. **Wittgenstein's philosophy of mathematics**. [S.l.]: Routledge, 2006.

FREGE, G. Os fundamentos da aritmética. Trad.: Luiz Henrique Lopes dos Santos. **Col.: Os Pensadores, Abril Cultural, São Paulo**, 1978.

GANDY, R. The Confluence of Ideas in 1936. In: **A Half-century Survey on The Universal Turing Machine**. New York, NY, USA: Oxford University Press, Inc., 1988. p. 55–111. ISBN 978-0-19-853741-0. Disponível em: <http://dl.acm.org/citation.cfm?id=57249.57252>.

HASELAGER, P.; GROOT, A. de; RAPPARD, H. V. Representationalism vs. anti-representationalism: a debate for the sake of appearance. **Philosophical psychology**, v. 16, n. 1, p. 5–24, 2003.

KANT, I. **Lectures on logic**. [S.l.]: Cambridge University Press, 2004.

LEGRIS, J. Between Calculus and Semantic Analysis: Symbolic Knowledge in the Origins of Mathematical Logic. **Symbolic Knowledge from Leibniz to Husserl**, p. 79–113, 2012.

LEME, R. R. et al. Solving Sokoban Optimally with Domain-Dependent Move Pruning. In: **Intelligent Systems (BRACIS), 2015 Brazilian Conference on**. [S.l.]: IEEE, 2015. p. 264–269.

MARION, M. **Wittgenstein, finitism, and the foundations of mathematics**. [S.l.]: Oxford University Press, 1998.

PICCININI, G. Computation Without Representation. **Philosophical Studies**, v. 137, n. 2, p. 205–241, 2008.

RESCORLA, M. Against Structuralist Theories of Computational Implementation. **The British Journal for the Philosophy of Science**, v. 64, n. 4, p. 681–707, dez. 2013. ISSN 0007-0882. Disponível em: <https://academic.oup.com/bjps/article/64/4/681/1517623>.

RODYCH, V. Wittgenstein's Philosophy of Mathematics. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Spring 2018. Metaphysics Research Lab, Stanford University, 2018. Disponível em: <https://plato.stanford.edu/archives/spr2018/entries/wittgenstein-mathematics/>.

SANTOS, L. H. L. d. Tractatus logico-philosophicus. **São Paulo: EDUSP**, 1993.

SECCO, G. D.; NOGUEZ, P. M. R. Operar e Exibir: Aspectos do Conhecimento Simbólico na Filosofia Tractariana da Matemática. **Revista Portuguesa de Filosofia**, v. 73, n. 3/4, p. 1463–1492, 2017.

SHANNON, C. E. A mathematical theory of communication. **Bell system technical journal**, v. 27, n. 3, p. 379–423, 1948.

SPREVAK, M. Computation, individuation, and the received view on representation. **Studies in History and Philosophy of Science Part A**, v. 41, n. 3, p. 260–270, set. 2010. ISSN 0039-3681. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0039368110000403>.

TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. **Proceedings of the London mathematical society**, v. 2, n. 1, p. 230–265, 1937.

TURNER, R.; ANGIUS, N. The Philosophy of Computer Science. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Spring 2017. Metaphysics Research Lab, Stanford University, 2017. Disponível em: <https://plato.stanford.edu/archives/spr2017/entries/computer-science/>.

WEBER, R. F.; PôRTO, I. E. Introdução à Arquitetura de Computadores. **Porto Alegre, Instituto de Informática-UFRGS**, 1998.

WHITEHEAD, A. N. **A treatise on universal algebra: with applications**. [S.l.]: The University Press, 1898. v. 1.