Pontifícia Universidade Católica de São Paulo
PUC-SP

Ricardo Maciel Gazoni

Semiótica da programação: levantamento crítico e perspectivas peirceanas

MESTRADO EM TECNOLOGIAS DA INTELIGÊNCIA E DESIGN DIGITAL

São Paulo
2015

Pontifícia Universidade Católica de São Paulo
PUC-SP

Ricardo Maciel Gazoni

Semiótica da programação: levantamento crítico e perspectivas peirceanas

MESTRADO EM TECNOLOGIAS DA INTELIGÊNCIA E DESIGN DIGITAL

Dissertação apresentada à Banca Examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de MESTRE em Tecnologias da Inteligência e Design Digital: Aprendizagem e Semiótica Cognitiva, sob a orientação do Prof. Dr. Winfried Maximilian Nöth.

São Paulo
2015

BANCA EXAMINADORA

_____

_____

_____

Ad maiorem Dei gloriam

Agradecimentos

Agradeço a Deus por ter permitido a realização deste trabalho e a Ele dedico o resultado. Sua variada ajuda se deu, também, através de muitas pessoas:

- Cristiane, Felipe e Marcelo: obrigado por serem a minha vida, meu motor, meu fim e a razão d'eu ser.
- Jonathas Magalhães Pereira da Silva, cuja insistência de décadas e ajuda mais do que moral o tornam cúmplice de minha vida acadêmica.
- Fernando, irmão que, mais novo, sempre inspirou as perguntas, nunca fugiu das discussões —nem das brigas—, modelo a imitar.
- Professora Lucia Santaella, cujo apoio e incentivo foram decisivos, mas também: cujo trabalho tornou este possível. Só não lhe agradeço sua genialidade, esse mérito é de Deus.
- Professor e orientador Winfried Nöth, tranquilo, impassível, meticuloso: obrigado por seu tempo, sua paciência, sua sabedoria.
- Professor Jorge Albuquerque Vieira, por sua tão tocante, gentil e completa atuação, sobretudo dedicada ao conhecimento da linguagem do universo —um homem educado.
- Professor Ítalo Santiago Vega, obrigado por insistir na correção de certos conceitos, cresci muitíssimo com isso.
- Colegas Isabel Jungk, Adelino Gala, Eduardo Camargo: por todo o conhecimento, tempo, paciência e não nos esqueçamos dos artigos, livros, cópias, links, dicas de bibliografia e das mais que enriquecedoras conversas.
- Edna Conti, por sua segura presença.
- Inúmeras pessoas que não cabe nomear, mas que lembro; em especial às comunidades que desenvolvem software livre, sem o qual não só este trabalho, mas minha vida seria impossível.
- CAPES, cuja bolsa concedida contribuiu imensamente para a realização deste projeto.

"... é bom lembrar que cada uma das verdades científicas é devida à afinidade da alma humana com a alma do universo, por imperfeita que seja —é, sem dúvida— essa afinidade."

Charles Sanders Peirce (CP 5.47)

Resumo

O estudo examina os processos de programação do ponto de vista da semiótica. Com base em conceitos chave da computação e programação, apresenta numa análise crítica do livro "Semiotics of Programming" de Kumiko Tanaka-Ishii.
O ponto de partida é o argumento peirceano da afinidade entre as estruturas do raciocínio lógico e dos processos mecânicos nas máquinas lógicas de Jevons, Marquand e Babbage. O autor defende a tese de que a filosofia das máquinas desenvolvida por Peirce nesse contexto pode servir como base de uma semiótica computacional e ancorar uma semiótica da programação apropriada para ultrapassar abordagens preponderantes dualistas e cartesianas, conforme as quais o raciocínio humano difere essencialmente de processos na natureza em geral e de processos mecânicos em particular.
O estudo revela lacunas na teoria semiótica computacional de Tanaka-Ishii, propõe caminhos para preenchê-las através da aplicação da semiótica de Peirce e postula que a abordagem peirceana promete uma melhor compreensão dos processos cognitivos envolvidos na computação e na programação.

Palavras-chaves: Semiótica computacional, Programação, Charles S. Peirce, Semiótica peirceana

Abstract

The study examines programming processes from the point of view of semiotics. Based on key concepts of computation and programming, it presents a critical review of the book "Semiotics of Programming" by Kumiko Tanaka-Ishii.
The starting point is the Peircean argument of the affinity between the structure of logical reasoning and the mechanical processes in the logical machines of Jevons, Marquand e Babbage. The author takes the view that the philosophy of machines developed by Peirce in this context can serve as the basis of a computational semiotics and anchor a semiotics of programming appropriate to overcome prevailing dualist and Cartesian approaches, according to which human reasoning differs essentially from processes in nature in general and from mechanical processes in particular.
The study reveals gaps in the semiotic theory of computation by Tanaka-Ishii, proposes ways to overcome them through the application of Peirce's semiotics and argues that the Peircean approach promises a better understanding of the cognitive processes involved in computation and programming.

Keywords: Computational semiotics, Programming, Charles S. Peirce, Peircean semiotics

Sumário

Lista de ilustrações

Lista de tabelas

1 Introdução

Computadores são ubíquos; dependemos de software como nunca. Paradoxalmente não existe atualmente uma metodologia para criação de software (JOHNSON; EKSTEDT; JACOBSON, 2012). Trata-se de um problema reconhecido desde 1968 e que tem se mostrado de difícil solução: a metodologia criada nos anos 1970 como uma resposta ao levantado nas conferências da OTAN sobre o tema (NAUR; RANDELL, 1969; BUXTON; RANDELL, 1970) —conhecida como "Metodologia Estruturada"— revelou-se capaz de resolver alguns problemas que hoje são considerados demasiadamente simples, e mesmo assim não deu conta de lidar satisfatoriamente com a continuidade da manutenção dos sistemas criados de acordo com seus preceitos.

A taxa de sucesso em desenvolvimento de software permanece baixa. Nos casos de sucesso o acerto é atribuído a fatores independentes da metodologia utilizada para desenvolver os sistemas: por exemplo, o talento individual dos membros da equipe de programação, ou a disponibilidade do usuário do sistema —que não conhece programação— para se envolver no processo de desenvolvimento (STANDISH GROUP, 1994, 2010, 2013). Não obstante, e até por causa disso, têm sido propostos inúmeros modos de desenvolvimento de sistemas, com fins comerciais ou não, que tentam abordagens mais consistentes para o problema. Muitas são abandonadas, como por exemplo a que chega a propor uma via linguística, procurando partir de textos com descrições em linguagem natural para produzir especificações técnicas de sistemas (ABBOTT, 1983; MORENO, 1997).

Por outro lado, a semiótica de Peirce tem se revelado um manancial inestimável na extração de estratégias metodológicas para leitura e análise de processos empíricos de signos (SANTAELLA, 2002, p. XIV). Ainda que pouco utilizada diretamente como instrumento de análise —sua imensa importância teórica chama mais a atenção dos estudiosos do que suas possibilidades de aplicação prática—, apresenta recursos que têm sido utilizados em muitos diferentes assuntos, como por exemplo publicidade, mídia, arte, literatura, e aos próprios computadores. A suspeita de que a utilização da semiótica é relevante para a ciência da computação já foi apontada por Nadin (2011). O início do seu artigo situa um estado de coisas que parece persistir:

> O tema deste artigo pode ser apresentado de maneira simples: semioticistas defendem que o conhecimento da semiótica é relevante para a ciência da computação. Se é assim, então por que os cientistas da computação, com raras exceções, continuam ignorando a semiótica? Podemos reformular a questão: pode a semiótica contribuir para o conhecimento e para a prática da computação? (NADIN, 2011, p. 89).

A aplicação da semiótica à analise de máquinas não é nova. Há inúmeros trabalhos a respeito de semiose de máquinas, em particular semiose computacional. O foco desses trabalhos está na análise das ações das próprias máquinas. Mesmo Peirce (1887) escreveu sobre algumas máquinas lógicas de seu tempo: aparelhos mecânicos nos quais o usuário configurava as premissas de um silogismo e, ao girar de uma alavanca, via a conclusão apontada pela máquina (ver 4.3.2).

Em menor número são as abordagens que tentam aplicar a semiótica à questão de desenvolvimento e programação de sistemas, utilizando diferentes correntes de pensamento sobre semiótica —um exemplo interessante é o uso da semiótica no desenvolvimento de sistemas inteligentes (cf. GUDWIN; QUEIROZ, 2007). Muitas das abordagens voltadas ao desenvolvimento de sistemas se dedicam à análise da interface humano-computador (HCI, para *Human Computer Interface*), que é considerada uma área "suave" ("*soft*") em distinção à área mais técnica da programação de computadores, considerada uma área "dura" ("*hard*") (ANDERSEN, 1992). Juntamente com Andersen (1992), acreditamos que a semiótica pode ser utilizada também para a análise da parte dura do desenvolvimento de sistemas. Alguns dos principais exemplos, além do analisado especificamente neste trabalho, são:

- O texto fundador de Zemanek (1966), apresentado na Conferência de Linguagens de Programação e Pragmática da ACM (*Association for Computing Machinery*) em 1965, que toma conceitos da semiótica de Morris e os aplica às linguagens de programação.
  Como é esperado, faz sua análise levando em conta as dimensões sintática, semântica e pragmática introduzidas por Morris (cf. NÖTH, 1990, p. 50), mas não sem antes lembrar que há mais aspectos a levar em conta na linguagem, como formalização, os conceitos de mundo-objeto, linguagem-objeto e metalinguagem, e aspectos descritivos e prescritivos (ZEMANEK, 1966, p. 140).
  Zemanek insiste na importância de formalizar nas linguagens de programação não só a sintaxe, mas também a semântica e, ao menos até certo ponto, a pragmática (ZEMANEK, 1966, p. 140); entende que o computador, por exigir uma linguagem sem ambiguidades e por poder ser programado para somente dizer (imprimir é o termo que usa) o que só pode ser claramente dito, pode incorporar um ideal que atribui ao Círculo de Viena, o que chamou de *motto* de Wittgenstein: "falar claramente ou não falar nada", num entendimento do *Tractatus Logico-Philosophicus* (WITTGENSTEIN, 2001) que pode ser questionado (cf. SANTOS, 2008).
  Destaca-se também em seu texto a defesa de dois tipos de pragmática: a humana e a mecânica, a segunda referindo-se à máquina em si. Mas o caráter determinístico de todas as etapas de transformação do programa em software encoraja o autor

a estender a pragmática mecânica a todo o processo de construção do software, a partir do programa.
Finalmente, seu texto traz *insights* que se revelaram premonitórios, como a importância na capacidade de apontar —apontar com o dedo ou outro dispositivo, no caso uma "caneta luminosa"— para transmitir informação ao computador, o que mais tarde se concretizou nos *mouse*, e à possibilidade então remota de levar o usuário humano a se comunicar através da simples escolha sucessiva de alternativas tipo sim/não, uma previsão do uso assistido de software.

- Liu (2000), que utiliza a semiótica para apresentar métodos para analise e modelagem de requisições. Sua abordagem é baseada na *semiótica organizacional* proposta por Stamper (1973), também ele propositor de formas para utilização da semiótica em sistemas de informação. A semiótica organizacional propõe seis níveis de análise dos signos, formando a "escada semiótica" (STAMPER et al., 2000, p. 12)

  1. Social
  2. Pragmático
  3. Semântico
  4. Sintático
  5. Empírico
  6. Físico

  De fato, de acordo com Gazendam (2004), Liu documentou e propôs a aplicação do método criado por Stamper e defendido por ele na força-tarefa conhecida como FRISCO ("*Framework of Information System Concepts*") da IFIP ("*International Federation for Informatin Processing*").

- Peter Bøgh Andersen foi o autor do primeiro livro sobre sobre semiótica computacional (GAZENDAM, 2004). Falecido em 2010, explorou a abordagem semiótica da computação em todos os níveis, sempre com uma grande preocupação com a aderência a evidências empíricas. Autodeclarava sua abordagem linguística como baseada no paradigma estruturalista europeu, rejeitando os paradigmas gerativos de Chomsky e o paradigma lógico de Carnap, Wittgenstein, Reichenach e Montague (ANDERSEN, 1992). Num equívoco que pode ser frequente, utiliza a "variante de Peirce do conceito de signo" pois nela "não faz sentido dizer que signos ocorrem exceto quando são interpretados por algum intérprete [...] produzindo um novo signo chamado de *interpretante*", o que, de acordo com o autor, corrige o modelo estruturalista ao introduzir a pessoa que interpreta o signo (ANDERSEN, 1992). Não obstante, sua abordagem traz *insights* interessantes quando utiliza a teoria

> da narração para definir os componentes dos sistemas, tanto na formação da interface quanto na determinação de partes do processamento (ANDERSEN, 1992; ANDERSEN, 1995).

A utilização da semiótica peirceana sem o conhecimento profundo dos fundamentos e implicações filosóficas de seus conceitos pode degenerar em uma "mera pirotecnia terminológica estéril" (SANTAELLA, 2002, p. XV). Em particular, Nöth (2014) nos lembra que há inúmeras teorias instrumentais dos signos —signos como instrumentos para alguma finalidade determinada por aqueles que os "usam"—, e afirma que essa ideia de instrumentalidade vai contra a teoria peirceana. O já citado artigo de Nadin (2011) revisa três livros cujos autores tentam aplicar a semiótica à ciência da computação: Liu (2000, reimpressão de 2005), Souza (2005) e Tanaka-Ishii (2010); a avaliação de Nadin: "se não fosse pela palavra semiótica —algumas vezes usada de modo mais que aproximativo— estes três livros não teriam nada em comum" (NADIN, 2011, p. 94). Ou seja, para Nadin a abordagem semiótica que esses cientistas da área da computação fazem é questionável, e isso parece ser a regra: "os três autores fundamentam-se em concepções completamente diferentes do que semiótica deve ser. Nesse sentido, eles representam toda a disciplina" (NADIN, 2011, p. 95).

A proposta do presente trabalho é analisar a utilização da semiótica peirceana na análise da programação de computadores, que é considerado um ramo "duro" da ciência da computação, tomando para isso o livro de Tanaka-Ishii, que também foi avaliado por Nadin. Assim, no Capítulo 2, *Contexto*, é feita uma pequena mas necessária definição dos sentidos que daremos a alguns termos da ciência da computação no presente trabalho. A seguir, no Capítulo 3, *"Semiotics of Programming"*, fazemos a leitura desse livro de Tanaka-Ishii (2010), com particular atenção à utilização da semiótica peirceana. O Capítulo 4, *Semiótica peirceana*, mostra os fundamentos e mecanismos propostos por Peirce em contraponto à visão chamada "cartesiana". Esse contraponto é importante, uma vez que grande parte da dificuldade de aplicação da semiótica peirceana à criação de sistemas parece decorrer de uma utilização pouco contextuada dos conceitos de Peirce. Segue-se, no Capítulo 5, *Semiótica de Peirce em "Semiotics of Programming"*, a análise da utilização da semiótica peirceana no livro; as considerações finais são apresentadas no Capítulo 6, *Conclusão*.

Esse estudo busca, assim, apontar caminhos possíveis para a utilização de semiótica de Peirce na programação de computadores através da compreensão do que já foi feito.

Nota sobre as citações da obra de Peirce

Como em inúmeros trabalhos, as citações da obra de Peirce que constam nos Collected Papers (PEIRCE, 1931-1958) serão acompanhadas do código "CP", seguido pelo número do volume e número do parágrafo.

2 Contexto

Durante este trabalho deparamo-nos com algumas dificuldades quanto a definições de termos relacionados à tecnologia da informação. Essas dificuldades parecem ser uma consequência natural da hodierna ubiquidade dos computadores: saindo dos laboratórios e das áreas técnicas das empresas, a terminologia antes acadêmica e destinada ao uso técnico ganhou novos espaços e com isto veio um esperado crescimento da informação veiculada pelos signos (cf. NÖTH; GURICK, 2011, cf. NÖTH, 2012). Não é sem consequências: por vezes geram expectativas em jovens que procuram profissionalização em tecnologia da informação esperando encontrar algo que é muito diferente do conteúdo acadêmico que efetivamente encontram (VEGA, 2014).

Julgamos necessário, então, expor o conjunto de definições que utilizaremos ao longo do resto do trabalho, esperando com isso também delinear com mais exatidão o alcance das definições acadêmicas. Além disso, faremos algumas considerações a respeito do desenvolvimento de sistemas e sua relação com a programação de computadores em 2.5.

2.1 Computador

O termo originalmente se referia a uma profissão: antes dos computadores eletrônicos, os computadores eram as pessoas que calculavam (computavam) números. Hoje em dia o termo traz outras evocações ao imaginário das pessoas. Por exemplo, uma busca na Wikipedia em inglês nos traz como definição "um dispositivo de propósito geral que pode ser programado para levar a cabo automaticamente um conjunto de operações aritméticas ou lógicas" (COMPUTER, 2014). A definição em português acrescenta: "pode possuir inúmeros atributos, dentre eles armazenamento de dados, processamento de dados, cálculo em grande escala, desenho industrial, tratamento de imagens gráficas, realidade virtual, entretenimento e cultura" (COMPUTADOR, 2014). Não são, evidentemente, definições acadêmicas. Academicamente, há interesse matemático nos chamados procedimentos computáveis —os procedimentos que seriam efetuados pelas pessoas que computavam, os ditos computadores; os famosos trabalhos de Church (1936) e Turing (1936) são exemplos (ver 2.2).

Mas uma questão importante é que o equipamento que usualmente chamamos de "computador" hoje em dia possui dispositivos internos e externos que o habilitam a realizar tarefas que não são computáveis —de acordo com a definição acadêmica do termo—, embora uma parte significativa das tarefas efetuadas por estes equipamentos

evidentemente o seja. Além disso, como se verá, a interpretação que hoje em dia se dá ao resultado de algumas das tarefas "estritamente computáveis" feitas por este equipamento ganhou dimensões muito diferentes das interpretações que eram dadas no início de sua utilização. Para uma pessoa no início do século XXI, quando se fala em computador o que vem à mente é um conceito que evidentemente carece da precisão acadêmica, mas que designa um equipamento que pode possuir:

- uma tela capaz de mostrar imagens em alta resolução,
- algum dispositivo para entrada de dados que pode ser um teclado, um *mouse*, a própria tela *touchscreen*, ou ainda outro: há inúmeros (*joystick*, *touchpad*...),
- alto-falantes, ou ao menos uma entrada para *headphones*,
- uma fonte de energia que pode ser uma bateria ou a própria rede elétrica,
- um ou mais dispositivos para conexão com outros computadores —via *internet* ou não,
- uma ou mais entradas em diversos padrões, para conexão de diferentes dispositivos; um padrão muito utilizado é o *USB* (*Universal Serial Bus*),
- algum recurso para que os dados permaneçam mesmo que o equipamento seja desligado —o "disco rígido" ou algo que o valha,
- uma certa quantidade de memória de trabalho (dita *RAM*),
- um processador que pode ter um ou mais núcleos com "velocidade" medida em gigahertz e
- muitos outros itens menos conhecidos, tais como por exemplo
  - um "acumulador de entropia" capaz de fornecer bits efetivamente aleatórios, e não pseudo-aleatórios como os que usualmente eram obtidos nesses equipamentos,
  - um ou mais acelerômetros capazes de detectar mudanças de posição do dispositivo,
  - uma bússola,
  - um localizador baseado em GPS (*Global Positioning System*),
  - um relógio.

Além de todos esses equipamentos, é subentendido que esse "computador" moderno possui algo chamado *sistema operacional*, que é o *software* mais básico do equipamento (v. *software* em 2.4).

Uma outra definição importante de "computador" —na verdade, "sistema automático de computar"— é a de von Neumann (1993, 1ª publicação em 1945). O relatório de von Neumann é considerado fundador pois delineou um circuito eletrônico com características que foram adotadas na construção dos equipamentos de computação automática a partir de então. A definição de von Neumann é dada a seguir:

> Um *sistema automático de computar* é um dispositivo (normalmente altamente complexo), que pode levar a cabo instruções para performar cálculos de considerável ordem de complexidade —por exemplo, resolver numericamente uma equação diferencial parcial não-linear em 2 ou 3 variáveis independentes (VON NEUMANN, 1993, §1.2).

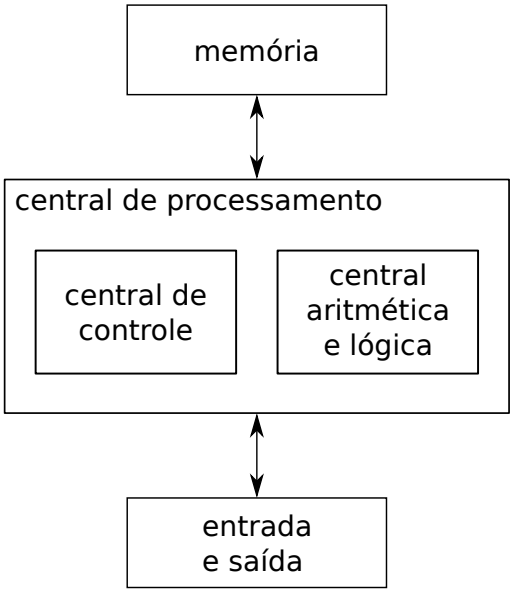
A principal característica é que, fazendo uso de "relés eletrônicos" (para os quais von Neumann sugeriu que fossem utilizadas as recém-inventadas válvulas), o circuito pode ter seu funcionamento alterado (ou seja, o circuito passa a se comportar como se fosse um outro circuito) conforme haja ou não certos estímulos elétricos nesses relés. Partindo dessa propriedade simples, e associando ao dispositivo um componente que emite um estímulo elétrico em intervalos regulares de tempo (chamado apropriadamente de relógio, em inglês *clock*), von Neumann especificou circuitos capazes de, por exemplo:

- armazenar ao longo do tempo grupos de estímulos elétricos que poderiam ser interpretados como números binários (*memória*),
- criar e apresentar grupos de estímulos elétricos —chamemo-los de "resultante"— a partir de outros grupos de estímulos elétricos —os "operandos"— tais que, interpretando-se todos esse grupos de estímulos como números binários, o número equivalente ao resultante é o resultado de um cálculo aritmético binário realizado entre os operandos,
- receber e emitir estímulos elétricos como forma de entrada e saída de informação.

O mais interessante é a central de controle. Através dela é possível controlar automaticamente o comportamento do circuito ao longo do tempo. Aproveitando-se da característica "mutante" do circuito —lembremos que seu comportamento se altera conforme a presença ou não de certos estímulos elétricos nas válvulas—, juntamente com os estímulos elétricos periódicos do *clock*, von Neumann foi capaz de definir um circuito eletrônico complexo cujo comportamento se altera automaticamente ao longo do tempo de acordo grupos de

estímulos elétricos armazenados nos próprios circuitos de memória. A Figura 1 esquematiza a disposição dos componentes em uma máquina com a arquitetura de von Neumann. Para

Figura 1 – Relações entre os componentes de uma máquina com a arquitetura de von Neumann (VON NEUMANN ARCHITECTURE, 2015, adaptado e traduzido)



manter o desenho o mais simples possível von Neumann definiu algumas restrições, como não permitir mais de uma operação simultânea e utilizar sempre o menor número possível de componentes.

Cumpre notar que von Neumann, neste texto de 1945, utiliza a palavra "código" ("*code*" em inglês) pela primeira vez ao dizer que "todos esses procedimentos [necessários para a realização dos cálculos preconizados] requerem o uso de algum *código* para expressar a definição lógica e algébrica do problema sob consideração, bem como o material numérico necessário" (VON NEUMANN, 1993, §1.2, destaque nosso). Aqui, portanto, "código" se refere a um modo de expressar definições lógicas, algébricas e numéricas através de recursos que não são as próprias definições em si. O que o autor propôs é um modo de interpretar séries de sinais elétricos presentes ou ausentes nesses elementos como entidades matemáticas, quer através de sua interpretação como números binários, quer através de uma tabela que associa cada sequência de estímulos a uma "ordem" como "somar", "subtrair" etc (VON NEUMANN, 1993, §15.6). O termo "código" é utilizado até hoje, sem precisão acadêmica, para se referir de maneira geral ao "conjunto de instruções executado pelo computador". "Codificar" nesse contexto significa "programar o computador".

A abordagem de von Neumann tem duas consequências que julgamos importante destacar:

- A correspondência entre as ordens matemáticas representadas pelo código e o comportamento dos circuitos elétricos feita por von Neumann abriu espaço para o tratamento puramente matemático do computador físico. Esse tratamento, embora enormemente útil para o desenvolvimento de melhores *softwares* (ver 2.4) negligencia aspectos físicos do computador; isso dificulta, por exemplo, o tratamento adequado de eventuais defeitos no equipamento.

- O fato de as ordens e os dados dos problemas compartilharem a mesma memória traz a vantagem de se poder manipular as instruções a serem realizadas do mesmo modo que os dados —o que é bom, já que ambos podem então ser manipulados pelas mesmas estruturas e de acordo com as mesmas regras— mas traz também a inesperada consequência de permitir que dados que não constituem instruções sejam tratados como tal, o que leva o equipamento a um comportamento errático.

Um fato curioso é que von Neumann usou o que então se sabia a respeito do funcionamento do sistema nervoso humano como inspiração para seu dispositivo; em seu relatório aparecem as palavras "neurônio", "sinapse" e "órgão", essa última como referência a partes do próprio dispositivo. Podemos agora definir a terminologia que usaremos doravante:

Computador ou computador estritamente von Neumann quando nos referirmos a um dispositivo eletrônico que possui a arquitetura de von Neumann e suas restrições. De modo resumido, é um aparelho eletrônico que contém dispositivos de entrada e saída e cujo comportamento se modifica automaticamente ao longo do tempo de acordo com estímulos elétricos colocados na memória. John von Neumann não definiu em seu relatório como seria o processo de entrada e saída do dispositivo, o mecanismo de registro e leitura dos dados permanentes, nem o procedimento pelo qual o dispositivo iniciaria a execução das ordens em sua memória. Esses pontos são obviamente endereçados em computadores modernos (v. a seguir), mas não há um modo único de fazê-lo; voltaremos oportunamente a estas questões.

Equipamento moderno ou computador moderno quando nos referirmos à definição mais ou menos imprecisa de computador usada hoje em dia, com todos os equipamentos auxiliares, como exposto a partir da página 24. Os computadores modernos contém em seu interior um ou mais dispositivos que se comportam estritamente como dispositivos de von Neumann.

Observamos que os computadores modernos não têm mais certas restrições impostas inicialmente por von Neumann, como por exemplo a restrição a uma operação por vez

ou a interpretação dos estímulos elétricos como informação binária de natureza numérica —sobre esta última afirmação podemos dizer que é evidente que uma sequência de estímulos elétricos gerada por um computador moderno sempre pode ser interpretada como uma sequência zeros e uns, que por sua vez pode ser interpretada como um número binário, mas os estímulos elétricos gerados pelos computadores modernos (e mesmo os gerados por suas partes que funcionam estritamente como dispositivos com arquitetura de von Neumann) frequentemente são utilizados diretamente para gerar imagens ou sons sem que haja necessidade de serem interpretados como números nesse processo.

2.2 Máquina de Turing

A máquina de Turing é um aparelho imaginário descrito por Alan Turing em 1936 (TURING, 1936) em um artigo sobre o problema da decisão (*Eintscheidunsgproblem*). A ideia e suas variações são largamente utilizadas nas ciências da computação; a título de exatidão traduziremos o texto original do artigo:

> Podemos comparar um homem no processo de computar um número real a uma máquina que é capaz apenas de um número finito de condições $q_1, q_2, \ldots, q_R$ que serão chamadas 'configurações-m'. A máquina é fornecida com uma 'fita' (o análogo ao papel) que corre através dela, e [é] dividida em seções (chamadas 'quadrados') cada uma capaz de carregar um 'símbolo'. A qualquer momento há somente um quadrado, digamos o r-ésimo, carregando o símbolo $\mathfrak{S}(r)$ o qual está 'na máquina'. Podemos chamar esse quadrado de 'quadrado escaneado'. O símbolo no quadrado escaneado pode ser chamado de 'símbolo escaneado'. O 'símbolo escaneado' é o único dos quais a máquina está, por assim dizer, 'diretamente consciente'. De qualquer modo, através da alteração de sua configuração-m a máquina pode efetivamente lembrar alguns dos símbolos que ela 'viu' (escaneou) previamente. O comportamento possível da máquina a qualquer momento é determinado pela configuração-m q_n e o símbolo escaneado $\mathfrak{S}(r)$. Este par $q_n, \mathfrak{S}(r)$ chamar-se-á 'configuração': assim, a configuração determina o possível comportamento da máquina. Em algumas das configurações nas quais o quadrado escaneado é branco (i.e. não carrega nenhum símbolo) a máquina escreve um novo símbolo no quadrado escaneado: em outras configurações ela apaga o símbolo escaneado. A máquina pode também trocar qual quadrado está sendo escaneado, mas apenas deslocando-o um lugar para a direita ou para a esquerda. Adicionalmente a qualquer uma dessas operações a configuração-m pode ser trocada. Alguns símbolos escritos formarão a sequencia de algarismos [*figures*, em inglês] que são a [representação] decimal do número [*number*, em inglês] que está sendo computado. Os outros são apenas notas de rascunho para 'auxiliar a memória'. Serão somente essas notas de rascunho que serão suscetíveis de serem apagadas.
>
> Afirmo que essas operações incluem todas as que são usadas na computação de um número (TURING, 1936, p. 231–232).

Convém notar que a demonstração da última afirmação depende, teoricamente, de uma definição formal do que é a "computação de um número". Mais à frente, no mesmo artigo,

Turing mostra que sua noção de computabilidade e a de "calculabilidade efetiva" de Church (1936) são equivalentes. Ambas as noções são tentativas de formalizar uma noção intuitiva de computação —ou função computável, o que vem a ser a mesma coisa pois o que se observa é que "todas as tentativas de se formalizar a noção intuitiva de função computável fornecem exatamente a mesma classe de funções" (CARNIELLI; EPSTEIN, 2006, p. 121). Podemos enunciar, então, o que ficou conhecido como tese de Church-Turing: "uma função é computável sse é computável por máquina de Turing" (CARNIELLI; EPSTEIN, 2006, p. 122). O "sse" no enunciado é lido como "se e somente se", e significa que a afirmação vale nos dois sentidos: uma função é computável se é computável por máquina de Turing e uma função é computável por máquina de Turing se é computável. Não pode ser formalmente provada já que há não definição formal para o conceito intuitivo de computável (a tese poderia ser tomada como uma definição do conceito). Entretanto pode ser refutada se for encontrada uma máquina teórica mais poderosa que a máquina de Turing —caso em que a nova máquina seria capaz de computar funções não computáveis na máquina de Turing.

Retomando o texto de Turing (1936), em seguida à descrição inicial o autor define o que é uma "máquina automática": é uma máquina em que a cada estágio o movimento é completamente determinado pela configuração. E chama esta máquina de "máquina-a", em oposição à "máquina-c" (*choice machine*, ou máquina de escolha) cujo movimento é apenas parcialmente determinado pela configuração: quando tal máquina chega a uma configuração ambígua ela não pode continuar até que alguma escolha arbitrária seja feita por um operador externo, que não precisa ser necessariamente um ser humano. Turing apenas apresenta o conceito de máquina de escolha; seu artigo trata somente de máquinas automáticas. Mais tarde, em sua tese de doutorado, Turing introduzirá um terceiro tipo de máquina, a "máquina-oráculo", que possui uma configuração adicional na qual a máquina "chama o oráculo" para dar um passo não-computável (HODGES, 2013). É importante destacar a diferença entre a máquina de escolha e a máquina-oráculo: ambas, em determinado momento, param aguardando uma decisão. A máquina de escolha só continua com o auxílio de um operador externo, mas o passo determinado pelo operador pode ser computável (ou seja, definível por uma outra máquina de Turing automática —lembremos que o operador não precisa ser humano). Já a máquina-oráculo só continua graças a uma decisão não-computável, ou seja, a decisão não pode ser tomada por uma outra máquina de Turing automática.

Adicionalmente, Turing (1936) define "máquina de computar", uma máquina automática que imprime dois tipos de símbolos, dos quais o primeiro tipo (chamados números – *figures* em inglês) consistem inteiramente de 0 e 1, os outros sendo chamados de "símbolos do segundo tipo"; suprida com uma fita em branco e colocada em movimento, começando da configuração-m correta, a sequência de símbolos impressas por ela que são do primeiro tipo chamar-se-á "sequência computada pela máquina". O número real cuja

expressão binária é dada pela introdução de um ponto decimal no início da sequência é chamado "número computado pela máquina". E introduz o conceito de "configuração completa": em qualquer estágio do movimento da máquina, a configuração completa é descrita pelo número do quadrado escaneado, pela sequência completa de todos os símbolos na fita, e pela configuração-m. As mudanças na máquina e na fita entre duas configurações completas sucessivas são chamados "movimentos" da máquina.

Finalmente Turing dá duas classificações possíveis para a máquina de computar: "circular" e "livre de circularidade"; as primeiras chegam a configurações a partir das quais não há movimento possível, ou se o movimento continua ela possivelmente continua imprimindo símbolos do segundo tipo mas não do primeiro tipo. A classificação é importante porque a definição de "sequência computável" é aquela que pode ser computada por uma máquina de computar livre de circularidade; e "número computável" é aquele que difere por um inteiro do número computável por uma máquina de computar livre de circularidade.

Portanto, "máquina de Turing", no artigo original, e levando em conta a teoria de Peirce a respeito do raciocínio diagramático (ver 4.3.1), parece ter a função de um diagrama imaginado e utilizado por Turing para deduzir propriedades matemáticas do ato "efetivo" de calcular. Em particular, é uma máquina automática de computar que Turing utiliza para descrever o que ficou conhecido como "máquina de Turing de computar universal", ou "máquina de Turing universal", uma máquina de Turing cujas configurações-m são tais que permitem à máquina universal comportar-se como qualquer outra máquina de Turing de computar cujas configurações-m estejam apropriadamente descritas na própria fita da máquina universal.

O exposto até o momento nos permite apresentar algumas outras importantes considerações:

- Pode-se traçar um paralelo entre a máquina de Turing universal e o dispositivo de von Neumann:
  - em ambos, há um conjunto fixo de configurações (configurações-m no caso da máquina de Turing, os próprios circuitos eletrônicos no caso do dispositivo de von Neumann) que
  - permitem que um comportamento descrito em outra parte da máquina (as configurações-m descritas na fita da máquina de Turing universal, as ordens na memória do dispositivo de von Neumann) seja o comportamento apresentado pela própria máquina.

De fato, Turing (1986) numa palestra dada em 1947 à London Mathematical Society, afirma mesmo que "máquinas como o ACE [*Automatic Computing Engine*, um computador digital no qual trabalhou] podem ser tomadas como versões práticas desse tipo de máquina [referindo-se à maquina de Turing descrita a partir da página 28 do presente trabalho]" (TURING, 1986). Ou seja, para Turing os computadores podem ser considerados como versões práticas da máquina que apresentou ao mundo.

- Turing nunca apresentou um tratamento matemático nem para as máquinas de escolha, nem para as máquinas-oráculo —essas últimas não poderiam sequer ser descritas como máquinas, de acordo com Turing (TURING, 1939 apud HODGES, 2013).

- Não obstante, o comportamento dos computadores modernos, se pode ser modelado como o comportamento de uma máquina de Turing, é o de uma máquina de Turing de escolha: em certo momento eles aguardam um operador externo para continuar realizando ações. Mesmo computadores com arquitetura estrita de von Neumann, conforme definido neste trabalho na pagina 27, poderiam ser programados para se comportar como máquinas de escolha dependendo de como fossem especificados itens não definidos no relatório de von Neumann, tais como os processo de entrada e saída, o mecanismo de registro e leitura dos dados permanentes e o procedimento pelo qual o dispositivo iniciaria a execução das "ordens" em sua memória. Essa é uma característica tem sido explorada inclusive para propor outros modelos de computação (cf. EBERBACH; GOLDIN; WEGNER, 2004).

Observa-se, portanto, que apesar de se tentar utilizar o modelo matemático legado por Turing para estudar o comportamento dos computadores modernos (conforme definição neste trabalho, na página 27), somente esse modelo não é suficiente para isso. Não foi sequer concebido para tal: seu objetivo, brilhantemente atingido, era o de estudar a computabilidade efetiva. Como terminologia, doravante, neste trabalho:

Máquina de Turing é a máquina correspondente à descrição de Turing, traduzida a partir da página 28. Os demais termos (máquina de Turing automática, máquina de Turing de computar, máquina de Turing universal, máquina de Turing de escolha) serão utilizados em seu próprio sentido. Outros conceitos relacionados (autômato finito, por exemplo) serão definidos se e quando forem necessários.

Computação é a atividade realizada por uma máquina de Turing de computar livre de circularidade.

2.3 Algoritmo

Há inúmeras definições para algoritmo. Alguns exemplos: "descrição finita de um procedimento. Conjunto de ações bem definidas que produzem um resultado pretendido após um tempo finito de processamento" (RAMOS; NETO; VEGA, 2009, p. 641 – glossário). "Procedimento sistemático que produz —num número finito de passos— a resposta a uma questão ou a solução de um problema" (ALGORITHM, 2014). Trata-se de uma ideia que padece da mesma dificuldade da ideia intuitiva de computabilidade: parece haver um consenso quanto ao que é um algoritmo, mas carecemos de uma definição formal adequada. O termo parece ter origem no século IX, quando o matemático muçulmano Mohammed ibn Musa al-Khowarizmi publicou um pequeno livro descrevendo como calcular utilizando um novo sistema numérico posicional de dez símbolos desenvolvido na Índia (EBERBACH; GOLDIN; WEGNER, 2004; FANT, 1993). Entretanto, muito antes disso a noção de algoritmo já era conhecida. Um exemplo é o algoritmo de Euclides, utilizado para calcular o maior divisor comum de dois números.

Convém observar que em torno desse conceito há dois conjuntos distintos: um conjunto de ações e um conjunto de instruções para performar as ações. Não raro ambos se confundem em algumas definições, como a já citada de Ramos, Neto e Vega (ao menos no seu glossário, ver acima). Para decidir a nomenclatura que utilizaremos, vejamos algumas considerações de dois autores: Knuth (1997) e Fant (1993). Comecemos pelas características desejáveis de um algoritmo segundo Knuth:

Finitude Um algoritmo deve sempre terminar após um número finito de passos. Observemos que Knuth se refere claramente ao conjunto de ações performadas: espera-se que essas ações terminem, e não continuem indefinidamente. É importante notar que a definição de computação —conforme a página 31 deste trabalho— não guarda essa característica: uma função computável pode não terminar num número finito de passos. O próprio Knuth nos dá um exemplo de método computacional que não termina ((KNUTH, 1997)), o que chamou de *processo reativo*, que continuamente interage com o ambiente —à maneira dos computadores modernos, embora nesse caso, aparentemente, trata-se de um processo semelhante ao de uma máquina de Turing de escolha e não o de uma máquina de Turing de computar.

Definição Cada passo do algoritmo precisa ser definido precisamente; as ações a serem levadas a cabo devem ser rigorosamente especificadas para cada caso de maneira inequívoca. Aqui o autor se refere claramente à descrição das instruções, lembrando-nos que uma vez que a língua corrente (no caso dele, inglês) pode ser ambígua, há linguagens formais, ditas de programação, que são "projetadas para especificar algoritmos, nas quais toda a afirmação tem um significado muito definido" (KNUTH,

1968, p. 5). Trataremos desse ponto mais adiante, e ao longo do presente trabalho.

Entrada Um algoritmo tem zero ou mais entradas: quantidades que são dadas a ele inicialmente antes do algoritmo "começar", ou dinamicamente, à medida que o algoritmo "roda" (é executado). E aqui, embora se esteja claramente falando das ações (só elas são executadas), lembremos que as instruções podem também, por exemplo, descrever quais entradas são esperadas.

Saída Um algoritmo tem uma ou mais saídas, "quantidades que têm uma relação específica com as entradas" (KNUTH, 1968, p. 5). Novamente, são as ações que nos dão as saídas, embora essas possam também ser descritas, ao menos em linhas gerais, nas instruções.

Efetividade "Espera-se em geral que um algoritmo seja efetivo, no sentido de que suas operações devem ser todas suficientemente básicas para que elas possam em princípio ser feitas exatamente e em um período finito de tempo por alguém usando papel e lápis" (KNUTH, 1968, p. 6). Aqui o autor decididamente mistura as ações às suas descrições: está dizendo que as descrições são tais que suas ações possam ser feitas "por alguém usando papel e lápis", e que isso caracteriza um algoritmo "efetivo".

Cabe aqui a consideração de que, aparentemente, para Knuth um algoritmo é uma descrição de ações tal que não deixa dúvidas quanto à ações que prescreve e que, além disso, descreve um conjunto de ações que termina gerando um resultado esperado. Knuth utiliza o conceito de algoritmo porque, como muitos, o julga fundamental para a programação de computadores; e continua o texto afirmando que não basta que um algoritmo tenha as características acima (como por exemplo a finitude: não basta ser finito, tem de ser finito num tempo razoável). Segundo ele, queremos, na prática, algoritmos que sejam "*bons* em algum senso estético frouxamente definido" (KNUTH, 1968, p. 7).

Não obstante, Fant (1993) critica fortemente a utilização do conceito matemático de algoritmo —conceito que, lembremos, carece de definição formal— em computadores modernos, em parte pelas razões expostas até aqui e que aparentemente tem a ver com o descasamento entre os três conceitos: de computação, de algoritmo, e aquilo que é efetivamente realizado por computadores modernos. Ele dá a seguinte definição de algoritmo como sendo "típica":

1. Um algoritmo deve ser uma sequência passo-a-passo de operações.
2. Cada operação deve ser precisamente definida.
3. Um algoritmo deve terminar num número finito de passos.

4. Um algoritmo deve efetivamente produzir uma solução correta.

5. Um algoritmo deve ser determinístico no sentido de que dada a mesma entrada ele vai sempre produzir a mesma solução.

Observemos que de novo há características aplicáveis às ações, e características aplicáveis às instruções. E Fant a seguir pergunta: "é fácil ver como esta lista de características restritivas serve para definir o que é aceitável como solução matemática, mas que serviço conceitual a noção de algoritmo presta à ciência da computação?" (FANT, 1993, p. 4). Justifica sua colocação dando exemplos pertinentes:

- um circuito lógico não é uma sequencia de operações;
- não se espera que um sistema operacional termine nem que chegue a uma solução;
- um sistema operacional não pode ser determinístico porque precisa se relacionar com entradas não coordenadas do mundo exterior;
- qualquer programa que utilize entradas aleatórias para levar a cabo seus processos —como simulações de Monte Carlo ou simulações de lógica *fuzzy*— não é um algoritmo;
- nenhum programa com erro pode ser considerado um algoritmo, embora seja geralmente aceito que não é possível demonstrar que programas relevantes sejam livres de erros;
- programas e computadores (no nosso sentido de computadores modernos) que utilizem processamento concorrente, nos quais muitas operações são levadas a cabo simultaneamente, não podem ser considerados algoritmos; finalmente, pergunta
- o que significa quando um programa sequencial que poderia ser considerado como um algoritmo é paralelizado por um compilador e não pode mais ser considerado um algoritmo? (Essa colocação a respeito do compilador tem a ver com o método utilizado para colocar ordens na memória tanto de computadores estritamente von Neumann como de computadores modernos, método que delinearemos a seguir).

Fant (1993) atribui ao trabalho de Markov publicado em 1954 a moderna definição de algoritmo do ponto de vista matemático: "em matemática, 'algoritmo' é comumente entendido como sendo um prescrição exata, definindo um processo computacional, levando de vários dados iniciais ao resultado desejado" (MARKOV, 1971 apud FANT, 1993, p. 3). Note-se que trata-se explicitamente de uma *descrição* que por sua vez "define um processo computacional". Se imaginarmos que Markov deu para "processo computacional" a mesma

definição de computação que adotamos neste trabalho, teremos que um algoritmo, para ele, pode determinar um conjunto infinito de ações.

Ainda que não haja consenso nos textos analisados quanto à utilidade do conceito de algoritmo para as ciências da computação, duas coisas parecem consensuais, e chamam a atenção:

- É esperado que num algoritmo o conjunto de descrições das ações conduza a um conjunto único de ações. Trata-se de uma "tradução perfeita". Ou: se a tradução não for perfeita, então não é um algoritmo. É uma conversão determinística: as ações são completamente determinadas pelas suas descrições.

- Um algoritmo tem um comportamento determinístico. Esse determinismo é caracterizado de diferentes maneiras. Markov o caracteriza atribuindo ao algoritmo a propriedade de ser computável. Essa característica é citada por Fant, mas Knuth não poderia caracterizar o determinismo utilizando a noção de computabilidade pois isso implicaria em que o conjunto de ações poderia ser infinito, o que contrariaria a característica da finitude que postula. Talvez por isso caracterize o determinismo de modo menos preciso, lançando mão de determinar as saídas de um algoritmo através de uma relação específica com suas entradas.

  Tomemos entretanto a instrução "lançar um dado e exibir o resultado". Embora possa ser uma instrução clara que leva a uma ação bem definida, não é uma instrução cabível em um algoritmo de acordo com as considerações de Markov e Fant, já que seu resultado não é computável. Já de acordo com Knuth não é possível dizer se essa instrução pode ou não pertencer a um algoritmo, já que a caracterização de algoritmo para esse autor não está presa à noção de computabilidade, embora sejamos inclinados a acreditar que para ele tal instrução também não poderia fazer parte de um algoritmo, uma vez que poderia ser considerada uma saída sem uma relação específica com alguma entrada —e aqui lembramos que não está claro o que o autor quer dizer com "relação específica". Podemos por exemplo imaginar uma situação em que o algoritmo trabalha de duas maneiras possíveis: se receber como entrada a palavra "estatística", apresenta o número de vezes em que foi executado. Se a entrada é a palavra "sorteio", sorteia um número de 1 a 6. Nesse caso, a instrução "lançar um dado e exibir o resultado", embora não tenha um resultado determinístico, tem como resultado uma saída que pode ser considerada como tendo relação específica com a entrada do algoritmo.

Após essas considerações, decidimos adotar a seguinte definição de algoritmo:

Algoritmo descrição que determina univocamente um conjunto de procedimentos que pode eventualmente, mas não necessariamente, ser realizado por uma máquina de Turing de computar livre de circularidades. Esse procedimentos têm, não obstante, um comportamento determinístico no sentido de que não é inesperado ou errático, sendo razoavelmente previsível. Notemos que a instrução "lançar um dado e exibir o resultado" tem um resultado cuja relação com as entradas do algoritmo pode não ser considerada específica, mas é ainda assim um resultado previsível: sabemos que pode ser um número entre 1 e 6, com chance de 1/6. A instrução pode, portanto, fazer parte de um algoritmo segundo nossa definição. Já a instrução "executar uma ação possível qualquer" não tem resultados previsíveis, não podendo fazer parte do algoritmo. Com isso temos consciência de estar adotando como definição de algoritmo uma ideia não formalizada que depende de uma visão subjetiva de previsibilidade.

Algoritmo finito Algoritmo cujas ações chegam a um estado final num número finito de movimentos.

2.4 Programa, linguagem de programação e software

Como se viu em Knuth (1997), os conceitos de programa e linguagem de programação às vezes são colocados a serviço da clareza do algoritmo. Vimos também em Fant (1993) que pode-se questionar a utilidade do conceito de algoritmo para a ciência da computação. Deve ficar claro que, assim como a noção de algoritmo, na qual uma descrição leva a um conjunto único de ações, levando à confusão dos dois conceitos —ação e descrição— num só termo, os programas de computador também superpõem em si diversas características:

Programas descrevem ações que serão performadas por computadores modernos: Do mesmo modo que nos algoritmos, a um programa corresponde, em linhas gerais, um único conjunto de ações. E, do mesmo modo que os algoritmos, programas podem descrever procedimentos que não são realizáveis por uma máquina de Turing de computar livre de circularidades, como, por exemplo, aguardar a digitação do usuário. Finalmente, nos computadores modernos (conforme definição da página 27), é possível que um programa determine uma ação equivalente a "lançar um dado e exibir o resultado". Mas também é possível criar um programa que determine uma ação equivalente a "executar uma ação possível qualquer", o que exclui os programas de computador da classe dos algoritmos: o comportamento de suas ações pode ser errático e imprevisível. Portanto, nem todo programa é um algoritmo.

Programas fazem parte do mecanismo que insere "ordens" no computador: Como vimos (ver 2.1), os computadores são o resultado de uma engenhosa combinação de

elementos eletrônicos em circuitos cujo comportamento varia de acordo com os estímulos elétricos aplicados a eles —as "ordens", segundo o próprio von Neumann. Já em seu relatório von Neumann apresenta uma tabela que enumera as ordens aceitáveis pelo dispositivo; conta com uma coluna com o "significado" da ordem escrito em termos de cálculo —por exemplo, "ordem para levar a cabo a operação w [uma entre adição, multiplicação etc] no CA [Controle Aritmético] e dispor do resultado"—, e uma outra coluna contendo um "símbolo curto" que em princípio poderia ser usado como mnemônico. Essa ideia evoluiu: assim que os computadores ganharam mais recursos começaram a surgir novas maneiras de descrever como deveriam se comportar. Essas descrições eram escritas segundo regras estritas, e continham textos cada vez mais semelhantes ao de procedimentos matemáticos e cada vez mais distantes do funcionamento interno da máquina. De fato, quanto mais distante das entranhas da máquina é a descrição utilizada, diz-se que ela é de mais "alto nível". Essas descrições são os programas de computador.
Nesse contexto os programas podem ser escritos com papel e lápis, e uma vez convertidos nos estímulos elétricos convenientes (através, por exemplo, de digitação num teclado) podem dar origem, através de um algoritmo finito computável, ao conjunto de estímulos elétricos que no momento adequado alterarão o comportamento dos circuitos elétricos do computador —as ordens de que nos fala von Neumann. Esse processo de criação de ordens a partir de programas pode ser realizado de diversas maneiras. Por exemplo, através de softwares específicos chamados de "compiladores".

Programas são cadeias válidas das linguagens de programação: Os programas são escritos em linguagens especiais, as linguagens de programação. O estudo das linguagens de programação distingue aspectos sintáticos e semânticos nelas. No aspecto semântico não há uma abordagem universalmente aceita (SEBESTA, 2012). Esse mesmo autor afirma que "se houvesse uma especificação semântica precisa para de uma linguagem de programação, poder-se-ia potencialmente provar que programas escritos na linguagem estão corretos sem necessidade de teste" (SEBESTA, 2012, p. 139). Sob o aspecto sintático, podemos pensar que o conjunto de caracteres nos quais podem ser interpretados os estímulos elétricos que fazem parte do programa são símbolos que formam um conjunto que chamaremos de "alfabeto". Concatenando elementos do alfabeto, produzimos "cadeias". "Linguagens formais" são um "conjunto, ou infinito, de cadeias de comprimento finito, formadas pela concatenação de elementos de um alfabeto finito e não vazio" (RAMOS; NETO; VEGA, 2009, p. 79). Dito isso, podemos definir as *linguagens de programação* como linguagens formais cujas cadeias são construídas de acordo com um conjunto finito de regras de formação (RAMOS; NETO; VEGA, 2009, p. 88 e 89). Programas são cadeias válidas das linguagens de programação.

Tendo em mente o colocado acima, podemos agora definir como usaremos esses

termos no presente trabalho:

Software Estímulos elétricos na memória do computador que determinarão o comportamento dos circuitos sujeitos a eles, e que serão impostos a esses circuitos ao longo do tempo. São evidentemente passíveis de ser interpretados como ordens a serem executadas pelo computador.

Linguagem de Programação Linguagem formal cujas cadeias são construídas de acordo com um conjunto finito de regras de formação, e que constituem os programas desta linguagem.

Programa Cadeias válidas da linguagem de programação que podem ser convertidas em software.

2.5 Desenvolvimento de sistemas

É curioso como diferentes ramos da ciência evoluem em diferentes velocidades. O desenvolvimento de grandes sistemas de software iniciou-se aproximadamente na década de sessenta — a disciplina "Engenharia de Software" ("Software Engineering") foi criada em 1968 (TELES, 2005). Na mesma época —mais precisamente em 3 de dezembro de 1967— foi realizado o primeiro transplante cardíaco. Mais de quarenta anos depois, no ano de 2009, somente 32% dos projetos de desenvolvimento de software aplicativo tiveram sucesso; 24% falharam completamente e 44% foram considerados comprometidos, apresentando atrasos, custos maiores que os previstos e/ou entregas inferiores às esperadas (STANDISH GROUP, 2010). No mesmo ano o prognóstico de êxito (entendido como sobrevida maior que cinco anos) em transplantes cardíacos era de 73,2% para homens e 69,0% para mulheres (HEART TRANSPLANTATION, 2014). Não custa lembrar que a alta taxa de mortalidade dos primeiros transplantes só começou a ser superada em meados dos anos 70, com a descoberta de imunodepressores capazes de lidar com o problema da rejeição, como a ciclosporina (COLUMBIA UNIVERSITY DEPARTMENT OF SURGERY, 2014). Já as taxas de acerto na produção de software mantém-se aproximadamente as mesmas há pelo menos dez anos (STANDISH GROUP, 2013), embora aparentemente tenha melhorado muito desde o início dos levantamentos feitos pelo Standish Group: em 1994 era de 16,8% (STANDISH GROUP, 1994).

2.5.1 A crise do software

"Crise do software" é um termo cunhado em 1968, quando se admitiu pela primeira vez uma crise latente na área (TELES, 2005). Entre 7 e 11 de Outubro de 1968 a OTAN patrocinou uma conferência intitulada "Engenharia de Software", com foco nas

questões básicas e principais problemas dessa matéria. O nome provocativo foi escolhido propositalmente por "implicar na necessidade de basear a manufatura de software nos mesmos tipos de fundamentos teóricos e disciplinas práticas que são tradicionais nos ramos estabelecidos da engenharia" (NAUR; RANDELL, 1969). Mas as diferenças entre a manufatura de software e as atividades das demais engenharias são tantas que chegam a dar margem a questionamentos acadêmicos quanto à propriedade do termo (BRYANT, 2000) que, não obstante, permanece. A crise parece ter emergido com a evolução natural da tecnologia: à medida em que os equipamentos ficavam mais potentes e baratos aumentava o desejo de utilizar plenamente tal potencial; o foco das atenções volta-se então para a programação —antes o foco era o equipamento— e o que se percebeu é que não havia prática de programação em larga escala. Na época previa-se que "até o final dos anos 70 seremos capazes de projetar e implantar o tipo de sistema que agora sobrecarrega nossa habilidade de programação", a uma fração do custo e livres de defeitos (DIJKSTRA, 1972).

As razões para essa dificuldade não foram claramente apontadas. Num influente artigo publicado em 1986, Brooks Jr. aponta quatro características que qualificou como essenciais (em contraponto a acidentais, como na filosofia aristotélica) nos sistemas de software e que impediriam ganhos de produtividade "de uma ordem de grandeza [ou seja, dez vezes mais] nos próximos dez anos": complexidade, conformidade, mutabilidade e invisibilidade (BROOKS JR., 1986). Cabe notar aqui que essas características se aplicam à "essência de uma entidade de software [que] é um construto de conceitos que se interligam [...]. Essa essência é abstrata, pois o construto conceitual é o mesmo sob muitas representações diferentes. Ele é, todavia, altamente preciso e ricamente detalhado" (BROOKS JR., 1986). O artigo revelou-se profético, a despeito das inúmeras réplicas que obteve, como o próprio autor publicou em uma avaliação dez anos depois e publicada no Brasil em 2009 (BROOKS JR., 2009). Outros autores acrescentam à lista de Brooks Jr.: "ausência de leis básicas", "imaturidade" (decorrente da rápida evolução tecnológica) e "baixo custo de manufatura" (TELES, 2005). Booch et al. (2007) acrescenta ainda que a complexidade do software derivaria de quatro elementos: "complexidade do domínio do problema", "dificuldade de gerenciar o processos de desenvolvimento", "flexibilidade possibilitada através do software" e "dificuldade de caracterizar o comportamento de sistemas discretos" (BOOCH et al., 2007).

2.5.2 Desenvolvimento de software e programação de computadores

Brooks Jr. acredita que a parte mais difícil na construção de software é a especificação. É nela que se vai determinar quais "construtos de conceitos" serão implementados. Uma vez determinados tais construtos, é teoricamente possível construir o software através da programação de computadores. A determinação dos construtos depende, naturalmente, da finalidade do software. Essa finalidade é determinada por quem solicita o software. A

"crise do software" é o fato de que um pequeno percentual do software produzido corresponde às necessidades e metas de quem o solicita de modo a satisfazer suas expectativas, em projetos que frequentemente atrasam e/ou custam mais do que o previsto (BOOCH et al., 2007, p. 3). Qual a melhor abordagem para superar essa crise é um tema em debate desde a segunda conferência da OTAN sobre o assunto, realizada em Roma em 1969 como consequência direta da conferência anterior. Já nessa conferência identificou-se uma "lacuna de comunicação" ("*communication gap*") entre os participantes nas diferentes áreas, que, segundo os editores, tornou-se o mais importante resultado da conferência (BUXTON; RANDELL, 1970).

De qualquer modo, a atividade de programar computadores é uma das muitas etapas da atividade mais ampla de desenvolver sistemas, e não se pode atribuir a crise do software somente à atividade de programação. A programação é a criação do programa pelo programador. Ao pensarmos no ato de programar podemos levar em conta o fato deste ato estar inserido num contexto em que o software deve satisfazer expectativas que não são sempre as do agente que cria o programa.

A função pretendida

A crise do software mostra uma dificuldade bem definida: dado um desejo, uma "função pretendida" para um software, a taxa de sucesso na construção de um software que efetivamente satisfaça esse desejo, ou que execute a função pretendida, é baixa. Pode haver ainda uma outra dificuldade: dado um programa, como descobrir qual função ele efetua —passo essencial para descobrir se ele executa ou não a função pretendida. Já em 1969 Hoare publicou um artigo sobre o tema. No artigo explora as fundações lógicas da programação de computadores, e inicia afirmando —como era de se esperar— que "uma das mais importantes propriedades de um programa é se ele leva a cabo ou não a função pretendida com ele" (HOARE, 1969, p. 577). Para ele função pretendida pelo programa pode ser especificada como "asserções gerais a respeito dos valores que variáveis relevantes vão ter *depois* da execução do programa" (HOARE, 1969, p. 577). Ou seja, o programa reflete as intenções pretendidas com ele se (1) estas puderem ser descritas rigorosamente através de asserções a respeito dos valores de certas variáveis do programa (HOARE, 1969, p. 579) e (2) essas asserções forem confirmadas pelos valores atingidos pelas variáveis do programa.

O artigo mostrou que é possível determinar um conjunto de axiomas e regras de inferência subjacentes ao raciocínio a respeito de programas de computador, conclusão estendida por trabalhos posteriores (cf. KOZEN, 2000).

Paradigmas de programação

Estudos buscaram encontrar correlação entre as dificuldades na criação de software e as linguagens de programação utilizadas. Descobriu-se que não há grande correlação com a linguagem em si, mas com o que se chamou de "paradigma de programação" em que se baseia a linguagem. Os paradigmas de programação são modelos de referência que permitem abstrair o funcionamento do software a partir do programa. A lista a seguir foi baseada em (VEGA, 2014, 25.fev) e Sebesta (2012); convém observar que há hoje em dia linguagens que incorporam características de mais de um paradigma de programação:

Imperativo nesse paradigma a linguagem dispõe de instruções: ordens que serão obedecidas pelo computador. É, de acordo com Sebesta (2012), decorrente da implantação de computadores na arquitetura de von Neumann (1993). Ainda segundo Sebesta (ibid., p. 19), nesse paradigma as variáveis[1] são recursos centrais porque modelam as células de memória. Trata-se de um paradigma importante que permite identificar algumas subdivisões das quais as principais são:

- **Procedimental** nesse paradigma a sequência de instruções do programa corresponde à sequência de instruções do software.
- **Orientado a objetos** aqui o programa descreve um conjunto de objetos, com suas características e como respondem a mensagens. A execução do software é vista como a troca de mensagens entre esses objetos.

Funcional onde a execução do software é vista como a aplicação de funções a parâmetros. Esse paradigma difere fundamentalmente do paradigma imperativo porque as funções definidas matematicamente têm comportamento independente do contexto: têm sempre o mesmo resultado dados os mesmos parâmetros —uma característica conhecida como *transparência referencial*. E essa independência do contexto dispensa o programador do controle sobre o uso da memória (que nos paradigmas imperativos de certa forma determina o contexto) e, portanto, do controle sobre as variáveis.

Declarativo aqui o programa é um conjunto de declarações no domínio do problema. A implementação da linguagem é tal que o software encontra no conjunto de declarações o subconjunto que determina a solução do problema.

[1] Mais sobre variáveis na página 49.

3 “Semiotics of Programming”

Passemos à exposição da obra analisada. Kumiko Tanaka-Ishii é hoje professora na Escola de Graduação e Faculdade de Ciência da Informação e Engenharia Elétrica na Universidade de Kyushu. Na época em que escreveu o livro, era professora associada do Departamento de Informática Criativa na Escola de Graduação em Ciência da Informação e Tecnologia na Universidade de Tóquio. Tem interesses em linguística computacional e processamento de linguagem natural, além de semiótica computacional. Sua motivação para escrever o livro iniciou-se em uma conversa em 2001, na qual recebeu a sugestão de aprofundar a relação entre linguagens de programação orientadas a objeto e as teorias semióticas de Peirce.

A obra é dividida em quatro partes, de acordo com os pontos de vista através dos quais, segundo a autora, os fundamentos da semiótica podem ser examinados (cf. TANAKA-ISHII, 2010, p. 6-7):

- Introdução
- Parte I. Modelos de Signos
- Parte II. Tipos de Signos e Conteúdo
- Parte III. Sistemas de Signos

Essas partes foram resumidas na conclusão do livro (p. 193–197). Passamos a expô-las, com especial atenção aos capítulos 2, no qual a autora expõe os signos computacionais que serão objeto de estudo no livro, e 3, no qual a autora expõe o seu entendimento dos modelos diádico e triádico de signo. Claro está que não concordamos necessariamente com as opiniões da autora. Algumas de nossas considerações estão em 5.

3.1 Introdução

Contém os capítulos 1 e 2 (p. 1–21). No capítulo 1 é feita uma breve introdução, na qual além de apresentar o livro sugere que seus resultados podem auxiliar no estudo da semiótica. O capítulo seguinte apresenta o conceito de programa e os programas exemplo que pretende utilizar ao longo do texto.

3.1.1 Capítulo 1 - Introdução

De acordo com a autora o tema do livro é "reconsiderar a reflexividade como a propriedade essencial de sistemas de signos" (TANAKA-ISHII, 2010, p. 1), entendendo reflexividade como a "capacidade de uma função ou de um sistema interpretar o que é produzido por ele mesmo" (p. 176). Segundo ela, "os problemas subjacentes às linguagens de programação são fundamentalmente relacionados à reflexividade, e não é exagero dizer que a história do desenvolvimento das linguagens de programação é a busca pela manipulação adequada da reflexividade". Seu livro tenta "considerar linguagens de programação a partir do ponto de vista mais amplo dos signos e sistemas de signos" (p. 1-2). Para isso aplica a "teoria semiótica ao campo da programação de computadores e sob essa luz leva em consideração as propriedades gerais dos signos e sistemas de signos através do conceito de reflexividade" (p. 193). De acordo com ela, tal empreitada destaca a diferença entre signos humanos e signos computacionais, e comparar linguagens humanas e computacionais como sistemas de signos permite enxergar diferenças fundamentais, que "o livro sugere, são governadas pela reflexividade. [...] Embora pessoas fiquem confusas, elas raramente se tornam incontroláveis por causa de uma auto-referência. Em contraste, para computadores, reflexividade é uma das causas mais frequentes de mau funcionamento" (p. 2–3)[1].

Tanaka-Ishii enumera as possíveis contribuições da semiótica à computação e vice-versa para em seguida explicar que a estrutura da parte principal do seu livro tem correspondência com os pontos de vista de acordo com os quais os fundamentos da semiótica podem ser examinados: primeiro, através de modelos de signos, em seguida de acordo com os tipos de signos e o conteúdo que representam e finalmente de acordo com os sistemas que são construídos através dos signos. Prossegue observando que é uma estrutura diferente da estrutura dos estudos de linguagem em geral —que são, segundo ela, organizados de acordo com os domínios da "sintaxe, semântica e pragmática"— justificando sua escolha porque "a abordagem do livro origina-se da semiótica, que está situada no nível mais fundamental da linguagem, antes mesmo de se levar em consideração elementos da estrutura linguística tais como a sintaxe". Afirma a seguir que nesse sentido o significado de "linguagem" na livro é em sua "forma mais abstrata, referindo-se a um tipo de sistema de signos nos quais os signos são elementos linguísticos. Em outras palavras, eu considero uma linguagem como uma relação entre elementos linguísticos e suas interpretações" (TANAKA-ISHII, 2010, p. 7).

Tanaka-Ishii prossegue informando que a ausência de uma introdução teórica da

[1] Embora nas linguagens de paradigma funcional a iteração seja implementada através da recursão (SEBESTA, 2012, p. 676), que é uma forma de auto-referência; nesse caso a reflexividade não é causa de mau funcionamento, ao contrário: é necessária para o funcionamento. A autora trata esse tema com mais detalhes no capítulo 4 de seu livro.

semiótica em seu livro se deve ao fato de que não tomou "simplesmente uma teoria estabelecida por um semioticista e a aplicou aos programas de computador". De acordo com ela, "a teoria semiótica não estava suficientemente estabelecida para ser diretamente aplicada em uma forma completa que pudesse ser introduzida no começo do livro" (p. 7). Esclarece: "a maior parte dos capítulos no livro trata de algum problema semiótico que julgo fundamental, e o problema é analisado e hipoteticamente resolvido através de alguma adaptação da teoria semiótica através de sua aplicação a programas de computador. Essas conclusões hipotéticas de fato se aplicam, no sentido mais rigoroso, somente a programas de computador". Finalmente evidencia uma diferença que enxerga entre a teoria semiótica e a teoria da programação: esta última tem um corpo de conhecimentos rigoroso: "para linguagens de programação refiro-me apenas a teorias e conceitos já existentes no domínio da programação de computadores e as utilizo meramente para análise semiótica: uma vez que uma linguagem de programação é bem-formada e rigorosa, a teoria relevante é fundamentalmente clara" (p. 6–8).

3.1.2 Capítulo 2 - Signos computacionais em programas

Nesse capítulo apresenta seu conceito de linguagem de programação e programa:

> Uma linguagem de programação (ou computacional) é uma linguagem artificial projetada para controlar computadores e máquinas. Um texto escrito numa linguagem de programação é chamado de programa, e assim máquinas são controladas usando programas. Linguagens de programação seguem regras estritas de sintaxe e semântica, e um programador deve seguir essas regras para gerar um programa com o comportamento esperado. Uma vez escrito, o texto do programa é analisado sintaticamente, otimizado ou compilado se necessário, dependendo da linguagem, e então é executado, ou rodado, em máquinas (TANAKA-ISHII, 2010, p. 10).

Observamos que, numa prática aparentemente comum na literatura, o texto leva a crer que é o programa em si que é executado, e não o software gerado a partir ele.

Nesse capítulo mostra também dois programas exemplo que serão utilizados ao longo do livro: um escrito em Java, uma linguagem que segue o paradigma orientado a objetos, e outro em Haskell, linguagem que segue o paradigma funcional. Ambos os programas dão origem a softwares que chegam ao mesmo resultado, a saber, calcular a área de três formas geométricas simples. O programa em Haskell está na Figura 2. O programa em Java, na Figura 3. Nessas figuras destacamos em negrito os textos que correspondem a variáveis ou identificadores, destaque que não aparece nas figuras apresentadas no livro; tomamos essa liberdade porque veremos logo adiante que o livro dá importância especial às variáveis e identificadores em programas de computador.

Figura 2 – Programa exemplo em Haskell, adaptado de Tanaka-Ishii (2010, p. 12)

```
 1: data Shape = Rectangle  Double Double
 2:              | Ellipse   Double Double
 3:              | Circle    Double
 4:
 5: area (Rectangle widht height) = width*height
 6: area (Ellipse width height) = pi*width*height/4.0
 7: area (Circle radius) = area(Ellipse (radius*2.0) (radius*2.0))
 8:
 9: main = let
10:           r = Rectangle 5.0 8.0
11:           u = Ellipse 3.0 4.0
12:           v = Circle 3.0
13:           ss = [r, u, v]
14:          in
15:           for (\s -> putStr("area : "++show (area s)++"\n")) ss
```

O trabalho inicia dizendo que as formas estão representadas nos programas por uma modelagem suficiente para o propósito de cálculo da área: retângulo e elipse pela altura e largura, círculo pelo raio. Essas grandezas (altura, largura e raio) são representadas por decimais.

Prossegue colocando que no programa em Haskell distinguem-se três blocos: o primeiro da linha 1 à linha 3, o segundo nas linhas 5, 6 e 7 e o último da linha 9 até a linha 15:

- O primeiro bloco define a estrutura de dados para as três formas (retângulo e elipse por dois decimais, círculo por um somente: decimais denotados pela palavra '`Double`'), e os três são representados pelo tipo '`Shape`' (que quer dizer *forma* em português).
- O segundo bloco define três funções, cada uma delas representando um procedimento de cálculo para obter a área do tipo de forma correspondente. Observa-se que o cálculo da área do círculo é baseado no cálculo da área da elipse: o círculo é tomado como uma elipse na qual a altura e a largura são iguais ao dobro do raio.
- Finalmente o terceiro bloco ilustra o uso dos dados e funções definidos; é uma expressão do tipo '`let ... in ...`', que tem por sua vez dois blocos, e que significa que o segundo bloco (depois do '`in`', na linha 15) deve ser calculado nas condições definidas no primeiro bloco (após o '`let`' e antes do '`in`', nas linhas 10 a 13). Assim:
  - O primeiro bloco da expressão '`let ... in ...`' especifica as formas do retângulo, da elipse e do círculo que serão efetivamente calculadas, representadas pelos signo '`r`', '`u`' e '`v`' respectivamente; e porque o cálculo para essas formas

consiste na aplicação da função '`area`' definida nas linhas 5–7, as formas são colocadas numa *lista*, uma estrutura de dados desenhada para manipular múltiplos itens de dados um após o outro, representada pelos signo '`ss`' que será utilizado no bloco seguinte.

– No segundo bloco da expressão '`let ... in ...`', na linha 15, cada forma é extraída da lista e provisoriamente representada pelo signo '`s`'. Para cada forma, um texto contendo a palavra '`area: `' seguido pela representação decimal do número calculado é impresso, seguido por um caractere de controle (representado por '`"\n"`') que indica uma nova linha.

O programa, portanto, consiste numa parte em que são feitas definições, e outra parte em que essas definições são usadas, e isso ocorre tanto globalmente quanto localmente. Globalmente, os dois primeiros blocos representam definições e o terceiro, uso. Localmente, o primeiro bloco da expressão '`let ... in ...`' define os quatro signos '`r`', '`u`', '`v`' e '`ss`', enquanto o segundo bloco lhes dá uso. Finalmente a autora informa que as definições são um tipo de *declaração* (em inglês, "*statement*"), "a unidade básica de execução em um programa de computador" ao passo que o uso é descrito através de uma *expressão* (em inglês, "*expression*"); segundo ela, "toda expressão tem um valor, o que não é necessariamente requerido para uma declaração. Por exemplo, '`3`', '`4`', '`2*(3+4)`', e '`pi*width*height/4.0`' (na linha 7) são todas expressões", lembrando que definições e expressões são inter-relacionadas: "uma definição contém uma expressão (do lado direito do '`=`') e uma expressão pode incluir definições, como no primeiro bloco da expressão '`let ... in ...`'" (TANAKA-ISHII, 2010, p. 13).

O trabalho prossegue com o programa correspondente em Java, na Figura 3, que tem uma estrutura composta por cinco blocos, os quatro primeiros contendo a palavra '`class`', que definem as estruturas de dados para para os quatro tipos de forma: '`Shape`', '`Rectangle`', '`Ellipse`' e '`Circle`':

- A classe '`Shape`' primeiro declara '`width`' e '`height`' como decimais (através da palavra '`double`'), e declara duas funções, '`Shape`' e '`area`'. A primeira ('`Shape`', na linha 3) é necessária para a construção inicial de uma *instância* de dados de forma; a função '`area`' calcula a área.

  '`Shape`' é um tipo de dado representando as outras três formas: retângulos, elipses e círculos são, todos, formas. A mesma relação aparece entre as classes '`Ellipse`' e '`Circle`': um círculo é um tipo de elipse. Considerando essas relações entre os tipos de forma do ponto de vista de conjuntos matemáticos, as classes se relacionam através da palavra '`extends`'. Uma classe que estende outra *herda* as propriedades desta, e propriedades herdadas podem ser usadas sem declaração.

Figura 3 – Programa exemplo em Java, adaptado de Tanaka-Ishii (2010, p. 14)

```
 1: abstract class Shape {
 2:  double width, height
 3:  Shape(double w, double h) { width=w; height=h; }
 4:  public double area() {return width*height; }
 5: }
 6:
 7: class Rectangle extends Shape {
 8:  Rectangle(double w, double h) { super(w,h); }
 9: }
10:
11: class Ellipse extends Shape {
12:  Ellipse(double w, double h) { super(w,h); }
13:  public double area() {return Math.PI*width*height/4.0; }
14: }
15:
16: class Circle extends Ellipse {
17:  Circle(double r) { super(r*2.0,r*2.0); }
18: }
19:
20: void run() {
21:  Rectangle r = new Rectangle(5.0,8.0);
22:  Ellipse   u = new Ellipse(3.0,4.0);
23:  Circle    v = new Circle(3.0);
24:
25:  Shape[] ss = new Shape[] {r, u, v};
26:  for (Shape s : ss) {putStr("area: " + s.area() + "\n"); }
27: }
```

- A classe '`Rectangle`' que estende a classe '`Shape`' (na linha 7) e herda suas propriedades, a saber, '`width`' e '`height`' declaradas na linha 2 e a função '`area`' na linha 4[2]. Por isso a classe '`Rectangle`' possui tanto '`width`' e '`height`' quanto '`area`', embora não estejam declaradas na classe. A função '`area`' declarada na linha 4 determina o comportamento padrão da função em objetos que estendem a classe '`Shape`' (em termos técnicos, é o comportamento "*default*"). Como essa é a forma de cálculo da área do retângulo, a classe '`Rectangle`' não requer que essa função seja redefinida, o que não ocorre com as demais formas.

- A classe '`Ellipse`' redefine a função '`area`' na linha 13.

- A classe '`Circle`' herda a função '`area`' da classe '`Ellipse`'. É interessante aqui observar que a única diferença inteoduzida na class '`Circle`' é a forma como é construída, como uma elipse de altura e largura iguais ao dobro do raio.

[2] No livro há também uma nota de rodapé explicando que a função de inicialização é tratada de forma diferenciada em Java embora outras linguagens orientadas a objeto as tratem da mesma forma.

O trabalho enfim identifica o quinto e último bloco do programa como aquele em que acontece o uso das definições feitas nos quatro primeiros, de uma maneira semelhante ao programa em Haskell, ou seja, através de definições e usos locais (TANAKA-ISHII, 2010, p. 14–15).

Tendo explicado a estrutura dos programas, mostra os quatro tipos de "signo" que, segundo ela, aparecem em um programa de computador:

Literais são sequências de dígitos formando números ou cadeias de caracteres constantes.

Operadores são símbolos especiais como "+" (soma), "∗" (multiplicação), parênteses, aspas etc.

Palavras reservadas são palavras especiais que fazem parte do projeto da linguagem de programação.

Variáveis ou identificadores são palavras determinadas pelos programadores para uso dentro do programa. Uma variável ou identificador representa uma estrutura de dados e/ou uma função. Não podem ser iguais às palavras reservadas ou aos operadores.

A autora esclarece que o principal alvo da sua análise são os identificadores, que segundo ela "cobrem a maioria dos signos em computadores", já que ícones visuais e sons são definidos primeiro como identificadores em programas antes de serem usados (TANAKA-ISHII, 2010, p. 18), numa visão que nos lembra o trabalho de Hoare (1969). Curiosamente a autora não entra em detalhes quanto à diferença do papel das variáveis entre os paradigmas funcional e orientado a objetos, como apresentado na página 41, em *Paradigmas de programação*. E continua identificando quatro "níveis semânticos" dos identificadores, e em que capítulos serão estudados (p. 18–20):

- Nível de hardware do computador: um identificador representa um endereço de memória e um valor em bits armazenado nesse endereço na memória do computador. A alocação e gerenciamento da memória é uma preocupação que os programadores não precisam ter em muitas linguagens de programação, que lidam automaticamente com esse assunto, "muito estudado na teoria dos compiladores". Esse nível semântico é estudado no capítulo 10 de seu livro.

- Nível da linguagem de programação: segundo a autora, todos os identificadores são definidos e utilizados dentro de um programa, e esta definição e uso estipulam o que o signo é. O foco em termos de nível semântico em todos os capítulos, exceto nos capítulos 6 e 10, é este nível semântico de programação. Além de definição e uso, a

autora identifica duas outras camadas de interpretação nesse nível semântico, que são estudadas no capítulo 6 do livro:

  - camada de tipo, no qual é definido o tipo do identificador —se um inteiro, um número que pode ser fracionário ou um texto, por exemplo, e
  - camada do endereço, que, interpretado de acordo com o tipo de identificador, resulta no valor do identificador, que é seu significado no contexto do programa.

- Nível de linguagem natural: os programas são lidos e alterados por diversas pessoas e contém indicações sobre sua natureza através de comentários em linguagem natural e do uso de identificadores cujos nomes reflitam o seu papel no programa, facilitando a interpretação deste por outros programadores. Esse nível de interpretação a autora considera sujeito à análise pela semiótica das linguagens naturais, não fazendo parte do livro.

É interessante notar que ao determinar o nível de hardware como um dos "níveis semânticos" do identificador o texto induz à confusão entre o representado no programa e o que acontece no interior da máquina através do software. Embora seja verdadeiro que um identificador num programa leve ao que pode ser interpretado como uma alocação de um endereço de memória e o armazenamento de bits no interior desse endereço pelo software, não é possível somente a partir do programa determinar, por exemplo, qual endereço será alocado nem quais bits serão armazenados. Esses são elementos determinados justamente pelos compiladores, e dependem, entre outros fatores, do equipamento no qual o software será executado. Além disso, outras ações decorrentes da execução do software que também poderiam ser interpretadas como alocação e armazenamento de bits na memória não são determinadas por identificadores nos programas.

O capítulo termina com uma consideração ao que a autora chama de "visão pansemiótica". De acordo com ela, tomar a interpretação no nível da linguagem de programação significa interpretar os signos dentro do sistema semiótico, dispensando entidades externas tais como os objetos físicos que os programas representam. Ela chama esse ponto de vista (que dispensa elementos externos à interpretação) de "visão pansemiótica" e a atribui a Charles Peirce, com sua "ideia drástica de que 'o fato de que todo pensamento é um signo, tomado em conjunção com o fato de que a vida é um trem de pensamento, prova que o homem é um signo' (CP 5.314 [1868])" (TANAKA-ISHII, 2010, p. 20). De acordo com ela, ainda que não se concorde com a premissa de que todos os pensamentos sejam signos, ela adota a visão pansemiótica no livro porque permite a comparação de computadores com seres humanos no mesmo nível do sistema de signos. Termina: "O mundo da computação é um raro caso no qual a premissa básica da filosofia pansemiótica vale" (p. 21).

3.2 Parte I. Modelos de Signos

Dos capítulos 3 a 5, investiga modelos de signos, que no capítulo 3 é caracterizada como "confusão babilônica". Aqui o livro apresenta os dois principais modelos de signo, o de Saussure (que chama de "diádico") e o de Peirce ("triádico"), associando o primeiro à programação funcional e o segundo à programação orientada a objeto; e infere um modelo de correspondência entre os dois modelos de signo. No capítulo 4, para estudar o papel do significado, utiliza o cálculo λ (lambda) de Alonzo Church para demonstrar o que entende com sendo o "significante" e o "significado" nessas expressões —que podem ser auto-referenciais. Examina no capítulo 5 a "ontologia dos sistemas de signos", sugerindo que podem ser baseadas em "ser" (orientado a objetos) e em "fazer" (orientado a funções). Conclui no capítulo que o modelo triádico se aplica a ambas e o diádico somente ao "ser".

3.2.1 Capítulo 3 - A confusão babilônica

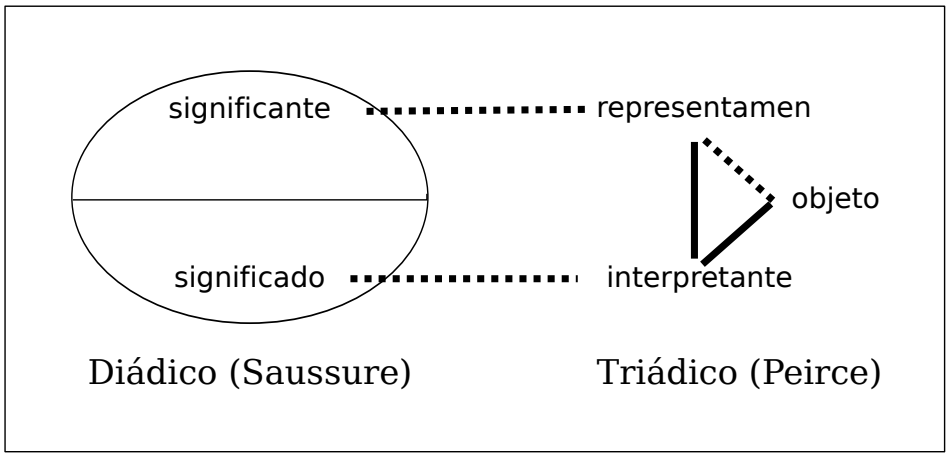
O entendimento que a autora tem do conceito de signo é exposto nesse capítulo, intitulado "*The Babylonian Confusion*" ("A Confusão Babilônica", citando Nöth (1990), p. 93). Fortemente baseada no "*Handbook of Semiotics*"[3] (NÖTH, 1990), apresenta dois modelos de signo —o diádico e o triádico— em sua evolução histórica. Mostra-nos as primeiras definições de signo para cada modelo —"*aliquid stat pro aliquo*", "algo que está em lugar de algo" para o modelo diádico, e a ideia de "objeto real $\leftrightarrow$ ideia $\leftrightarrow$ rótulo" para o modelo triádico— e ao final escolhe as nomenclaturas determinadas pelos modelos de Saussure (diádico) e Peirce (triádico) para desenvolver o tema do capítulo, que é "estabelecer uma hipótese para resolver essa confusão babilônica através da análise dos signos em programas de computador" (TANAKA-ISHII, 2010, p. 29). A autora pretende entender como os dois modelos são relacionados, pois tal entendimento "promove uma melhor compreensão de cada modelo e também das funções universais de cada um dos relatos [do signo em cada modelo]. Acima de tudo, leitores provenientes de um domínio relativamente formal duvidariam do impacto da semiótica se um conceito tão fundamental quanto a modelagem do signo for deixado indeterminado" (p. 28-29). Esse incômodo com a ausência de um modelo unificado parece motivar a autora, também ela proveniente de um domínio "relativamente formal".

Prossegue apresentando as hipóteses nas quais se pode estabelecer relações entre o modelo diádico e o triádico, assumindo que em ambos os modelos o signo em si ("rótulo" segundo ela) tem a mesma "dimensão", que corresponde ao significante em Saussure e o representamen em Peirce. Os relatos restantes teriam correspondência de acordo com uma de três hipóteses: o significado em Saussure corresponde ao objeto em Peirce; ou

[3] Trata-se de uma obra que almeja o "objetivo aventureiro de um levantamento topográfico das principais áreas de semiótica teórica e aplicada" (NÖTH, 1990, p. ix). Um texto, em nossa opinião, mais adequado à orientação de pesquisas adicionais do que à compreensão conclusiva do assunto.

ao interpretante; ou a ambos. De acordo com ela, numa hipótese "tradicional", tanto Umberto Eco quanto —incorretamente, em nossa opinião— Winfried Nöth (citando Nöth (1990), p. 59) indicam que a correspondência entre os dois modelos é a de que o significado em Saussure corresponde ao interpretante em Peirce, e que o objeto em Peirce não tem correspondência no modelo de Saussure. Essa conclusão é reforçada por uma citação de Eco, que afirma que "objetos não são levados em conta na linguística de Saussure" (ECO, 1979 apud TANAKA-ISHII, 2010, p. 30). O trabalho ilustra a hipótese tradicional como na Figura 4.

Figura 4 – Relacionamento entre os modelos diádico e triádico, conforme Tanaka-Ishii (2010, p. 31)



A seguir a autora apresenta uma "nova" hipótese, na qual se vale da distinção entre o objeto dinâmico e o objeto imediato em Peirce. Para ela, o objeto imediato corresponde à representação mental de uma entidade no mundo real, entidade essa que seria o objeto dinâmico. Então, propõe que a correspondência entre os dois modelos se dê de forma que o significado no modelo de Saussure corresponda ao objeto imediato no modelo de Peirce, e o significante em Saussure corresponda ao representamen em Peirce. Para determinar o papel do interpretante de Peirce na nova teoria, a autora se aprofunda na semiótica peirceana para descobrir que o interpretante tem um papel crucial na produção da semiose, que é "chamar outros signos que evocam interpretantes, que chamam outros signos, e assim por diante, levando a semiose infinita" (TANAKA-ISHII, 2010, p. 33). Citando Peirce (CP 8.184, 1909), afirma que "Peirce explica o interpretante com respeito ao termo *interpretação*" e assim "o interpretante serve para evocar a interpretação de signos na forma de semiose, e esta função está incluída no modelo de signo de Peirce". E o que precisa ser esclarecido a seguir, segundo o texto, é como tal interpretação se situa na filosofia de Saussure; e aqui chega ao conceito saussureano de *diferença*, para concluir que para Saussure os significados (*meaning*) dos signos existem no sistema total da linguagem, e, portanto, é "provável que no modelo de Saussure o uso do signo não esteja incorporado no

modelo de signo mas existe como um valor holístico dentro do sistema" (TANAKA-ISHII, 2010, p. 33). E, finalmente, para verificar se sua hipótese é mais acurada que a tradicional, propõe que analisemos o caso concreto dos signos computacionais.

O ponto que a autora quer ilustrar é como os identificadores são usados em ambos os programas-exemplo, ilustrados pela Figura 2 na página 46 e pela Figura 3 na página 48, lembrando-nos que o nível semântico dos identificadores é considerado na obra em termos de sua definição e uso dentro de um programa. Segundo ela, os dois programas são muito diferentes. No programa em Haskell (Figura 2) as funções para cálculo da área das formas são definidas fora das definições dos dados das formas, ao passo que no programa em Java (Figura 3) essas funções estão encapsuladas dentro de cada classe. Segundo o trabalho, a questão a ser endereçada é a localização do cálculo da área. No programa em Haskell, ela está localizada na função, separada da definição do dado, ao passo que no programa em Java esta definição pertence à classe de dados de cada forma. Prossegue observando que essa diferença aparece também no uso da função para calcular as áreas das formas. Na linha 15 da Figura 2 (programa em Haskell) esse uso é dado pela expressão '`(area s)`', ou seja, a função que toma cada uma das formas como argumento[4]. No programa em Java (Figura 3), esse uso aparece na linha 26 como '`s.area()`', e trata-se do uso de uma função —sem argumento— que pertence à classe '`s`'. A autora explica que no paradigma funcional (aqui representado pelo programa em Haskell) os programas são descritos por expressões funcionais, ou seja, funções —que são o mapeamento de um conjunto de entrada para um conjunto de saída. Aqui as principais entidades são as funções, e as definições de dados permanecem mínimas. Já no segundo programa, em Java, as principais entidades são objetos, ou seja, tanto dados quanto funcionalidades são empacotados juntos nas classes que instanciam os objetos. As definições de dados contém o máximo relacionado a eles, já que os cálculos são incorporados à definição.

A autora revela que essas formas de lidar com os identificadores os caracteriza de acordo com dois tipos: um, que chama de "diádico", e representa ("*stands for*") ou funções ou dados. O outro, chamado "triádico", representa tanto funções quanto dados. Na programação funcional, todos os identificadores são diádicos, ao passo que em programas que seguem o paradigma orientado a objetos aparecem identificadores tanto diádicos quanto triádicos. Dá exemplos deste caso no programa em Java (Figura 3), no qual os identificadores '`area`' nas linhas 4 e 13 são diádicos na medida em que representam apenas um procedimento, ao passo que os identificadores '`Shape`', '`Rectangle`', '`Ellipse`' e '`Circle`' nas linhas 1, 7, 11 e 16, respectivamente, os nomes das classes são triádicos uma vez que representam tanto dados quanto funcionalidades.

A relação entre os identificadores diádicos e triádicos e os respectivos modelos

[4] "Argumento" aqui é um termo técnico que significa "dado de entrada para a função".

de signo é apresentada no texto como segue, iniciando pela relação entre o paradigma funcional e o modelo diádico de signo:

> Um signo no modelo diádico tem um significante e um significado. Porque todos os identificadores diádicos consistem de um nome e seu conteúdo, o nome provavelmente corresponde ao significante e o conteúdo corresponde ao significado. Por exemplo, na Figura 2 o nome '`Rectangle`' é considerado um significante, representando as características dos dados de dois '`Double`'s, o significado. Através do nome '`Rectangle`' o conjunto de características (i.e., a largura e a altura) é coletado. Tal correspondência entre significante e significado também vale para identificadores que representam funções: o nome da função '`area`' é um significante tendo um procedimento de mapeamento como significado.
>
> Esse identificadores são usados por outros signos dentro do programa. Por exemplo, as formas representadas pelo signo '`s`' são usados pela função '`area`'. Notem que essa relação entre '`Rectangle`' e '`area`' não é conhecida a partir da posição do '`Rectangle`' (linha 1). Ao invés, é a função '`area`' que sabe que tem um '`Rectangle`' como argumento (linha 5). Esse fato adiciona significado [*meaning*, no original] ao signo '`Rectangle`'. Portanto, no paradigma funcional, identificadores adquirem significado [*meaning*] adicional pela forma como são usados, mas isso é externo à representação do identificador.
>
> Como na teoria de Saussure, então, diferença no uso tem um papel importante. [...] Em outras palavras, identificadores diádicos adquirem significado [*meaning*] a partir do uso, que é localizado externamente a seu conteúdo (TANAKA-ISHII, 2010, p. 36).

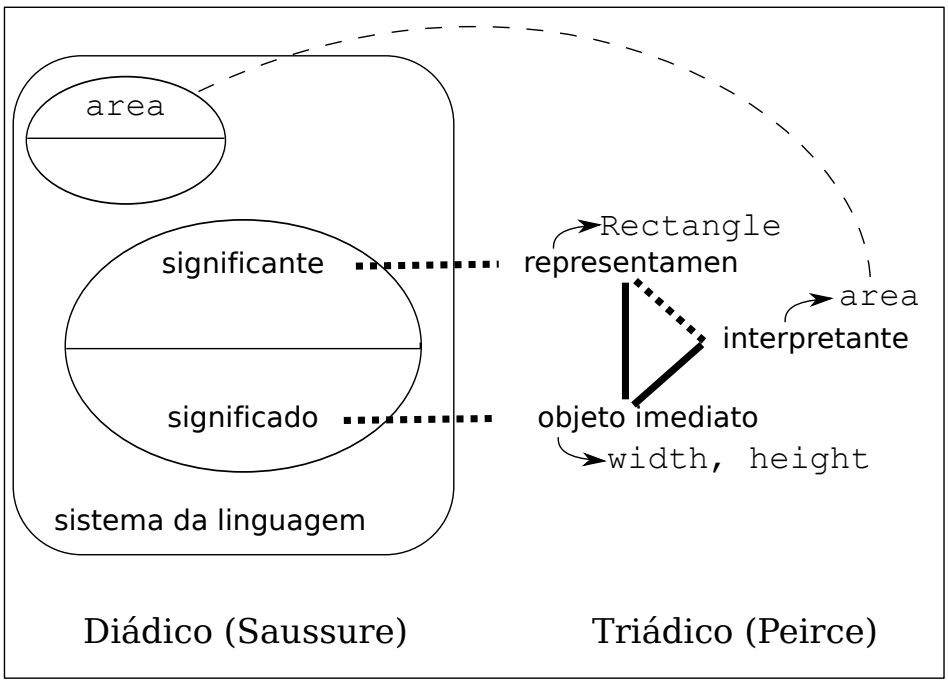
Um raciocínio semelhante é apresentado por ela para associar o paradigma orientado a objetos ao modelo triádico de signo:

> Um signo no modelo triádico tem um representamen, um objeto, e um interpretante. Uma vez que todos os identificadores triádicos no paradigma orientado a objetos consistem de um nome, dados e funcionalidades, estes se prestam à comparação com os relatos do modelo triádico de signo. Por exemplo, na Figura 3, um '`Shape`' como representamen representa dois '`double`'s (decimais), '`height`' e '`width`, e a função '`area`', que se aplica aos dados. A parte dos dados constitui o conceito do quê uma forma é, em termos de possuir uma largura e uma altura, então é a provável constituinte do objeto imediato na terminologia de Peirce. Essa parte não pode ser um interpretante porque não evoca nenhuma semiose. Ao invés, a semiose é evocada pelas funções que pertencem a cada classe. A função '`area`' utiliza multiplicação no seu cálculo, desse modo gerando um processo sígnico adicional. Além disso, a aplicação de uma função é análoga à interpretação porque a área do retângulo pode ser considerada como uma interpretação do retângulo. Assim, as funções definidas dentro de uma classe são consideradas interpretantes. Isso encaixa com a geração de semiose no modelo de Peirce, que requer apenas informação local incorporada dentro do modelo de signo e não requer todo o sistema da linguagem. O fato de que cada classe tem informação a respeito de sua funcionalidade difere do caso diádico, no qual é a função que sabe qual dado manipular (TANAKA-ISHII, 2010, p. 37).

Prossegue comparando o modelo diádico —no qual diferentes usos atribuem significado adicional aos identificadores diádicos— com o paradigma orientado a objetos, no qual o significado deve ser incorporado dentro da definição do identificador a partir do início: tudo o que adiciona significado a um identificador deve fazer parte de sua definição.

O capítulo termina com a autora revisitando a "confusão babilônica", na qual a correspondência entre os modelos de signo diádico e triádico é retomada à luz da discussão

Figura 5 – A nova hipótese de relacionamento entre os modelos diádico e triádico aplicada ao signo 'Rectangle', traduzido e adaptado de Tanaka-Ishii (2010, p. 39)



anterior. A correspondência proposta pela autora pode ser resumida nos seguintes pontos, conforme ilustrado na Figura 5:

- O significante em Saussure corresponde ao representamen em Peirce.
- O significado em Saussure corresponde ao objeto imediato em Peirce.
- O interpretante em Peirce está alocado no *uso* que aparece na diferença de Saussure como outro significado de um signo. No modelo de Peirce, o uso de um signo é representado como um interpretante, e a semiose é gerada pela chamada a um interpretante anexado ao próprio signo. No modelo de Saussure, de qualquer modo, a semiose é gerada por um signo sendo usado por um outro signo, que é usado por outro signo, e assim por diante.

O capítulo conclui afirmando que, na medida em que sua hipótese é correta, o modelo de Peirce é compatível com o de Saussure, e o modelo de Saussure pode ser obtido quando o interpretante de Peirce é localizado fora do modelo de signo mas dentro do sistema da linguagem. E consequentemente os modelos diádico e triádico são compatíveis, ambos aparecendo no sistema artificial de signos dos programas de computador. O sumário do capítulo entra em detalhes sobre a estrutura do livro, e apresenta a terminologia que usará ao longo do livro, apresentada na Tabela 1.

Tabela 1 – Nomenclatura de Tanaka-Ishii para os relatos do signo

Saussure (diádico)	significante	significado	valor holístico
Peirce (triádico)	representamen	objeto imediato	interpretante
Termos no livro	significante	conteúdo	uso

Fonte: traduzido e adaptado de Tanaka-Ishii (2010, p. 41)

3.2.2 Capítulo 4 - Casamento do significante e do significado

O capítulo se dedica à exploração adicional da "união entre o significante, conteúdo e uso definidos na seção (*sic*) anterior, especialmente pelo exame das distinções entre os relatos do signo" (TANAKA-ISHII, 2010, p. 45). E inicia com um detalhamento de como Saussure via a associação entre significante e significado.

O que a autora expõe é um aparente paradoxo entre a indissociabilidade entre significante e significado, a arbitrariedade da conexão entre ambos —que indica que não há necessidade essencial na ligação entre eles (p. 47)— e a noção de diferença, que dá origem ao valor do signo e que coloca o sistema de signos como um sistema de relações. Segundo ela, "as questões subjacentes a esse paradoxo são a associação entre significante e significado, se eles são ou não separáveis, e, acima de tudo, o *papel* de um significante com respeito a um significado" (p. 48). O objetivo do capítulo é considerar esse papel entre os signos computacionais, dentro do nível semântico da linguagem de programação (ver p. 49). Utiliza para isso o cálculo-λ, que introduz no livro, o que não faremos aqui, bastando-nos dizer que pode ser chamado de "menor linguagem de programação do mundo", constituindo-se de uma única regra de transformação (substituição de variáveis) e um único esquema de definição de função e, não obstante, é um formalismo capaz de expressar e calcular qualquer função computável, sendo equivalente à máquina de Turing (ROJAS, 1997) (subentendendo: máquina de Turing de computar livre de circularidade). Mais sobre a equivalência com máquinas de Turing pode ser visto em 2.2.

A autora introduz então o conceito de *articulação*, uma das duas diferentes funções

para a geração de signos (a outra é a nomeação): "*articular* neste livro significa construir uma unidade semiótica formada de signos. Por exemplo, o termo em linguagem natural 'frio' normalmente é estipulado pelas pessoas via temperatura relativa, mas pode também ser articulado por meio de sua definição em palavras, como no dicionário de inglês Oxford: 'a uma, ou tendo uma, baixa temperatura, especialmente quando comparado com o corpo humano' ". E afirma que os termos-λ podem ser considerados como uma articulação: $\lambda x.x+1$ (uma expressão-λ) articula o conceito complexo de adicionar um (TANAKA-ISHII, 2010, p. 53).

A seguir afirma que a inovação de Saussure foi elaborar um modelo de signo que nega o signo como nomenclatura, ou mero rótulo, deixando a articulação para o lado do conteúdo, e que situa a articulação no lado do significante, considerando os dois processos (nomeação e articulação) solidamente acoplados: dentro de qualquer modelo diádico depois de Saussure o significante, ou nome, é uma função para articular o significado; vale dizer: a articulação é performada pelo significante. E como o termo-λ tem as duas funções de articulação e nomeação, pode-se considerar que ele possui as funções de um signo, com uma diferença no que toca à nomeação: esta só ocorre na substituição de variáveis. A variável a ser substituída funciona como o significante do termo pelo qual ela é substituída, num signo em que este último termo é o significado. Assim, enquanto não há a substituição, o termo-λ funciona como um signo com significado, mas com significante em aberto. O processo de nomeação então é explicitado pelo calculo-λ, o que não ocorre no moderno modelo diádico de signo (p. 54–56).

A autora introduz, então, o conceito de definição no cálculo-λ (como uma extensão deste, chamando o novo cálculo de "cálculo-λ estendido") e através dele o conceito de auto-referência do signo. Com isso consegue criar definições auto-referentes dentro do cálculo-λ estendido. A autora também aponta que na auto-referência encontrada pelo cálculo-λ estendido nomeação e articulação ocorrem simultaneamente e, por causa disso, nos signos auto-referentes os modelos diádico e triádico se tornam equivalentes (p. 58–66).

Chama a atenção que a autora entenda que a auto-referência seja "mal adequada à computação" (p. 61). Na página seguinte lembra que o cálculo-λ e o cálculo-λ estendido são matematicamente equivalentes e cita uma demonstração de Church que mostra que toda função recursiva pode ser transformada em uma iteração, dispensando a auto-referencia (p. 62). Sabemos que a diferença é que nas funções auto-recursivas a iteração não é explícita (ver p. 44, rodapé). Portanto, exigem um raciocínio maior do programador para sua compreensão, mesmo dentro do nível semântico da programação. Esse ponto será melhor avaliado em 5.3.1.

Finalmente, citando uma prova teórica de que a equivalência de dois termos-λ não

pode ser julgada por um procedimento computável —ou seja, por um terceiro termo-λ—, afirma que tal julgamento através de um procedimento computável é frequentemente necessário em ciência da computação, o que é feito de maneira mais limitada através do julgamento da equivalência entre entradas e saídas: "Quando as entradas e saídas de duas expressões são diferentes, então elas são consideradas expressões diferentes". De acordo com o texto, "esse modo operacional de julgar a equivalência de duas expressões dentro do domínio da ciência da computação é análogo à declaração de Saussure: só há diferença entre signos. Esse fato sugere que tal dificuldade é inerente a sistemas de signos" (p. 67).

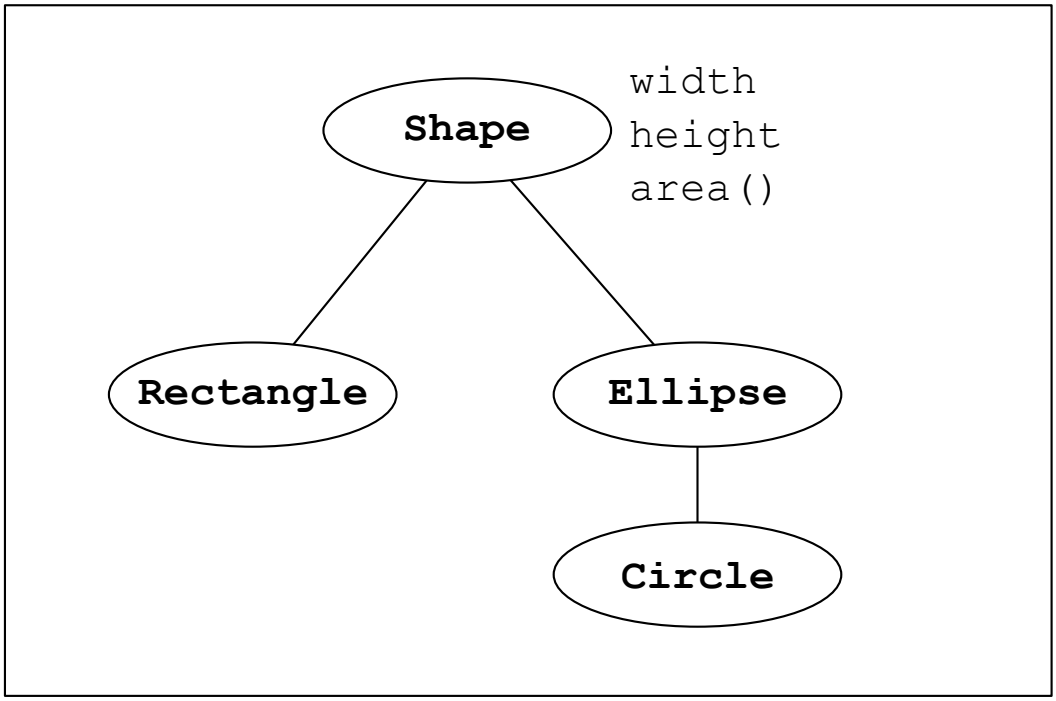
3.2.3 Capítulo 5 - Ser e fazer em programas

Este capítulo pretende mostrar "o grau em que os modelos de signos especificam o projeto de ontologia de um programa de computador", questionando uma "visão absoluta de ontologia", considerando "em que extensão uma ontologia é estipulada por um modelo sígnico" (TANAKA-ISHII, 2010, p. 69). O foco do capítulo é a "dicotomia ontológica entre 'ser' e 'fazer'. 'Ser' [...] refere-se ao estado ontológico de uma entidade cujo caráter ôntico é estabelecido pelo que ela é, enquanto 'fazer' denota [o estado ontológico] de uma entidade cujo caráter ôntico é especificado pelo que ela faz e pelo que pode ser feito a ela" (p. 71), um contraste que é especialmente proeminente, de acordo com ela, nas linguagens que seguem o paradigma orientado a objetos, o que sugere que a emergência da dicotomia é "resultado do modo pelo qual as entidades são descritas". A autora sustenta que uma ontologia de ou 'ser' ou 'fazer' devem surgir como resultado direto da "modelagem triádica da entidade: ênfase nas características levam a uma ontologia do 'ser', ao passo que uma ênfase nas funções leva a uma ontologia do 'fazer' " (p. 71). Ainda de acordo com o texto, as linguagens orientadas a objeto evoluíram de uma ontologia baseada no ser para uma mais próxima do fazer. As linguagens de paradigma funcional repousariam na ontologia do 'ser' (p. 71).

A explicação resumida de como um programa pode possuir signos com essa ou aquela ontologia inicia com a explicação de como o programador pode criar abstrações ao definir as entidades que farão parte do programa. Do ponto de vista técnico, essa diferença repousa no uso dos conceitos de "classe" e "tipo de dado abstrato", conceitos cujas minúcias técnicas tentaremos evitar tanto quanto possível.

Um programa que calcula a área de três figuras geométricas pode ser estruturado com base nas figuras geométricas em si, como foi visto no capítulo 2 do seu livro (corresponde ao programa em Java da Figura 3), cuja estrutura abstrata pode ser vista na Figura 6. A autora entende que esse objetos estão relacionados de acordo com uma ontologia do 'ser': um círculo *é* uma elipse, que por sua vez *é* uma forma, assim como um retângulo *é* uma forma; então a classe '`Shape`' é colocada como *mãe* das classes

Figura 6 – Uma hierarquia de objetos de formas simples: ontologia do 'ser', de acordo com Tanaka-Ishii (2010, p. 75)

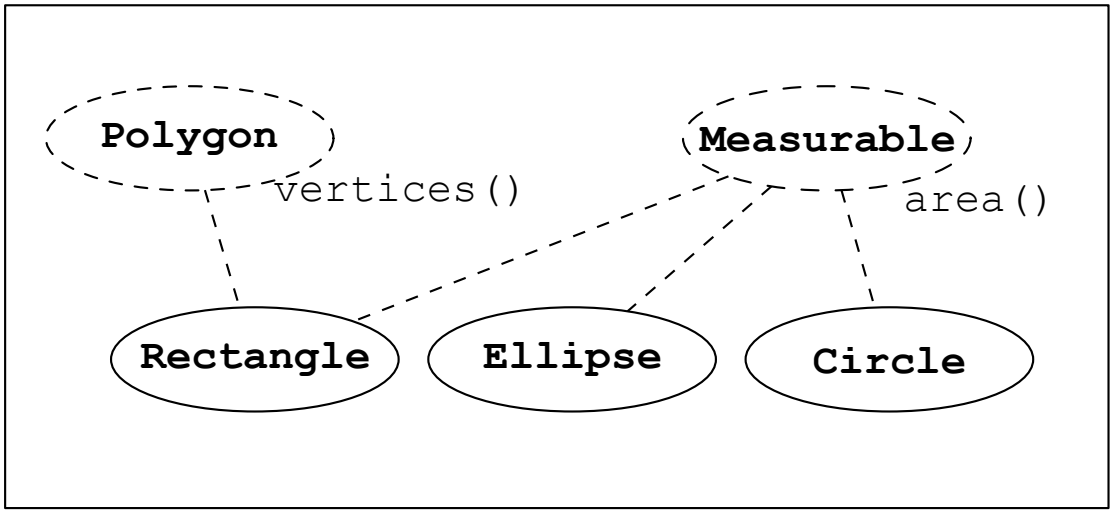


'`Rectangle`' e '`Ellipse`', que por sua vez é mãe da classe '`Circle`': numa relação de *herança* implementada pela palavra reservada '`extends`' no programa (que pode ser visto na Figura 3 na página 48), as classes filhas *herdam* características e funções de suas classes mãe (TANAKA-ISHII, 2010, p. 74–78). O trabalho não faz referência ao fato de que certas formas (que teoricamente pertenceriam à classe '`Shape`') não serem completamente descritos por sua altura e largura, como é o caso dos triângulos escalenos. Mesmo os círculos são melhor representados por seus raios que por suas alturas e larguras.

Já um programa que apresenta os mesmos resultados numa ontologia do 'fazer' pode ter sua estrutura apreciada na Figura 7. Nesse diagrama há objetos em linhas tracejadas: são as *interfaces*, implementadas como tipos de dados abstratos. As interfaces '`Polygon`' e '`Measurable`' declaram as funções pelas quais os objetos podem ser acessados, respectivamente '`vertices()`', que determina o número de vértices de um polígono, e '`area()`', que determina sua área. Somente a classe '`Rectangle`' implementa a função '`vertices()`', pois das três classes só ele é um polígono, mas as três classes implementam a função '`area()`'. Observa-se daí que são as classes que implementam as funções das interfaces, através da palavra reservada '`implements`' (no lugar de '`extends`') —as interfaces nunca implementam as funções que declaram. De acordo com a autora, não há aqui uma ontologia de 'ser' nas classes, mas de 'fazer' (TANAKA-ISHII, 2010, p. 78–81).

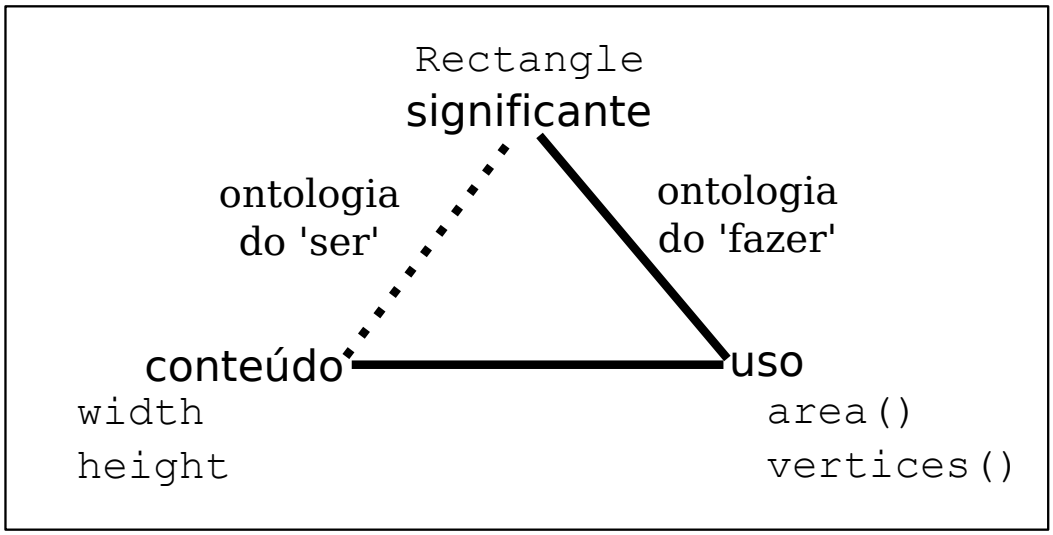
A autora associa as ontologias do 'ser' e do 'fazer' ao modelo triádico de signo de

Figura 7 – Uma hierarquia de objetos na ontologia do 'fazer', adaptada de Tanaka-Ishii (2010, p. 78)



acordo com a Figura 8. Nela a autora aponta a correspondência entre o uso e a ontologia

Figura 8 – O modelo triádico e as ontologias do 'ser' e do 'fazer', traduzido e adaptado de Tanaka-Ishii (2010, p. 82)



do 'fazer', e o conteúdo e a ontologia do 'ser'. E conclui que a diferença ontológica entre 'ser' e 'fazer' emerge dependendo de qual lado do modelo triádico é enfatizado durante a construção da ontologia, "exatamente como quando Saussure e Peirce reverteram 'aliquid stat pro aliquo'. Antes de Peirce ou Saussure, signos eram pensados como sendo nomes anexados a objetos, mas Peirce e Saussure reverteram isso de modo que o significante define o conteúdo" (TANAKA-ISHII, 2010, p. 82).

Conclui o capitulo fazendo algumas considerações filosóficas: entendendo que "esta diferença no foco da construção das relações [entre unidades de conteúdo –'ser'–, ou entre

usos –'fazer'] entre 'ser' e 'fazer' [...] deriva de uma diferença entre tomar um objeto de um ponto de vista interior ['ser'] ou exterior ['fazer']" (p. 83), mostra que, pelo fato de Peirce considerar o objeto no signo com secundidade, e sabendo que a primeiridade para Peirce é primordial para a secundidade e esta primordial para a terceiridade, "Peirce considerava seu objeto como sendo mais primordial que o seu interpretante. Isto revela que Peirce tomava os objetos do ponto de vista interior" (p. 84). Prossegue se perguntando se há pensamento na área de humanidades que corresponda ao 'fazer', tomando os objetos do ponto de vista exterior, e chega à conclusão de que Heidegger o faz (p. 84).

3.3 Parte II. Tipos de Signos e Conteúdo

A segunda parte contém os capítulos de 6 a 8 e ultrapassa a questão da modelagem dos signos, versando sobre tipos de signos e conteúdo. O capítulo 6 vai lidar explicitamente com os níveis semânticos de tipo e endereço, dentro do nível semântico de programação (como já explicado na página 49 deste trabalho). Para isso a autora vai utilizar conceitos de conotação e denotação de Hjelmslev, e os conceitos de ícone, índice e símbolo de Peirce. No capítulo 7 a autora pretende utilizar conceitos utilizados na computação, como "currying" e a transformação de Church para melhor compreender o conceito de terceiridade em Peirce. O capítulo 8 faz considerações a respeito da hecceidade[5] no contexto da computação, onde a informação digital é perfeitamente reprodutível e todas as instâncias de algo digital são literalmente idênticas entre si.

3.3.1 Capítulo 6 - A afirmação '`x := x + 1`'

O capítulo se dedica aos "diferentes níveis de representação sígnica para conteúdo [ou seja, a relação entre o significante e o que a autora chamou de conteúdo]" (TANAKA-ISHII, 2010, p. 94). Para isso expõe a afirmação que dá título ao capítulo, esclarecendo que não se trata de uma definição auto-referencial, mas uma expressão que incrementa de '`1`' o valor de '`x`', e que os dois signos '`x`' na afirmação não têm a mesma significação (p. 94).

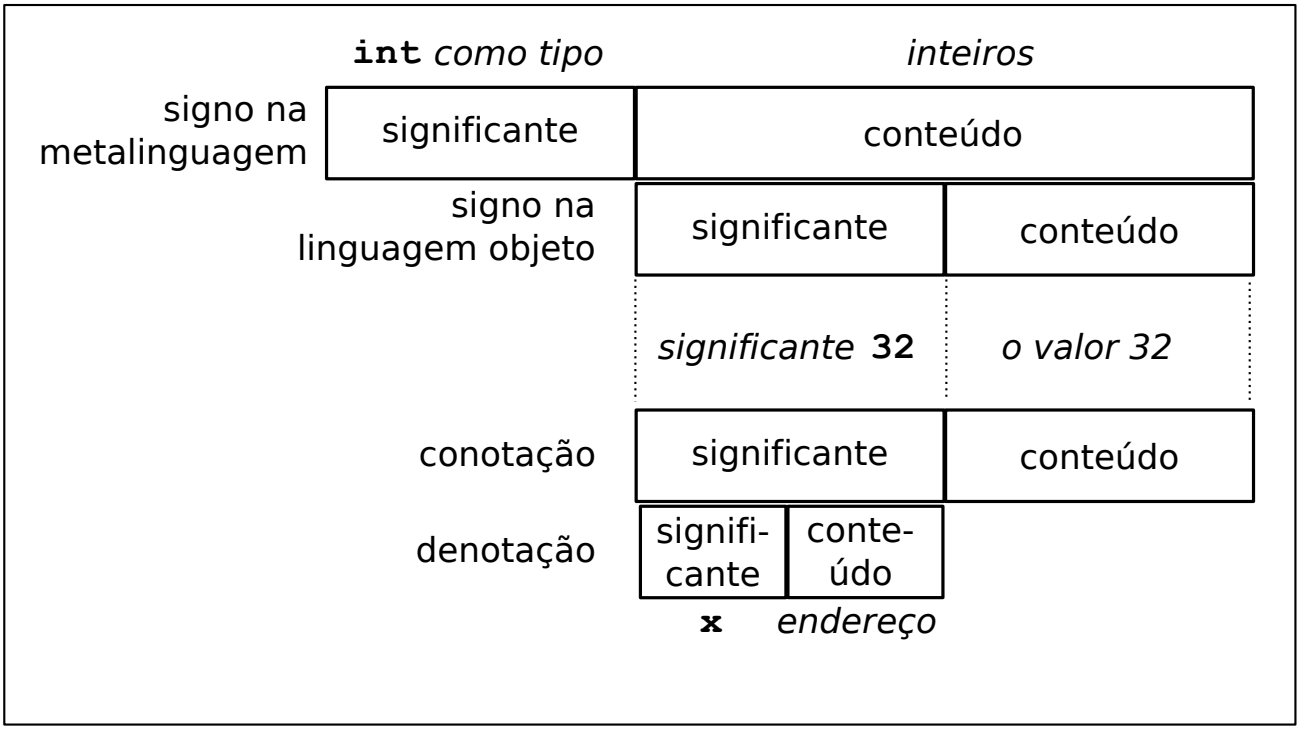
Segundo a autora, a ambiguidade na afirmação decorre daquilo que os identificadores realmente são, e não pode ser evitada (p. 95). Prossegue afirmando que os identificadores são usados de duas formas: como endereços na memória do computador, e como os valores armazenados nesse endereço. E tal ambiguidade é resolvida de acordo com o contexto. Na afirmação dada ('`x := x + 1`'), o contexto é o lado do operador de atribuição '`:=`': segundo ela, tipicamente o '`x`' do lado esquerdo indica um endereço, ao passo que o '`x`' do lado direito indica o valor (p. 96). Se a afirmação pudesse ser traduzida em instruções, seria: "coloque no endereço indicado pela variável '`x`' o valor da variável '`x`' acrescido de

[5] Aquilo que caracteriza algo como particular: "istude" ("*thisness*"), de "isto" + "tude".

1", resultando no incremento por '`1`' de '`x`'. O fato de o conteúdo (conjunto de bits) do endereço indicado por '`x`' ser interpretado como um número decorre, por exemplo, do fato de que em algum outro ponto anterior do programa o comando '`int x := 32`' ter sido executado, ao mesmo tempo declarando que os valores de '`x`' são números inteiros (graças à palavra reservada '`int`') e guardando no endereço apontado por '`x`' a representação em bits do número 32, o que é feito pelo texto '`32`' do comando.

Essas características dos identificadores são inicialmente analisadas dentro da moldura de conotação/denotação e metalinguagem de Hjelmslev, de acordo com a Figura 9. Nela se vê que o significante '`x`' denota um endereço de memória, mas conota o valor

Figura 9 – Identificadores numa moldura de conotação/denotação e metalinguagem, traduzido de Tanaka-Ishii (2010, p. 101)



32, denotado pelo significante '`32`'. E no nível da metalinguagem, o significante '`int`' tem como conteúdo os inteiros: o próprio valor 32, denotado pelo mesmo significante '`32`' (TANAKA-ISHII, 2010, p. 100–103).

Em seguida a autora analisa essas mesmas características dentro da moldura peirceana de ícone, índice e símbolo. Apresenta inicialmente as três categorias de Peirce (CP 8.328, 1904)—cuja significância reconsiderará no próximo capítulo:

Primeiridade é o modo de ser daquilo que é tal como é, positivamente e sem referência a nada mais.

Secundidade é o modo de ser daquilo que que tal como é, com respeito a um segundo mas independente de qualquer terceiro.

Terceiridade é o modo de ser daquilo que é tal como é, ao trazer um segundo e um terceiro em relação um com o outro.

Classifica-as "não como categorias de signos, mas categorias de formas e conteúdo" (TANAKA-ISHII, 2010, p. 104) e prossegue lembrando que de acordo com Peirce um signo é um "exemplo típico de uma forma de terceiridade" (p. 104) e que a classificação peirceana diz respeito às relações entre os signos e seus relatos. E prossegue: "uma vez que nosso foco é o nível no qual o conteúdo é representado no signo, examinamos a classificação do signo em respeito ao conteúdo", que Peirce (CP 2.275, 1903) considerava a mais fundamental divisão dos signos: ícone, índice e símbolo.

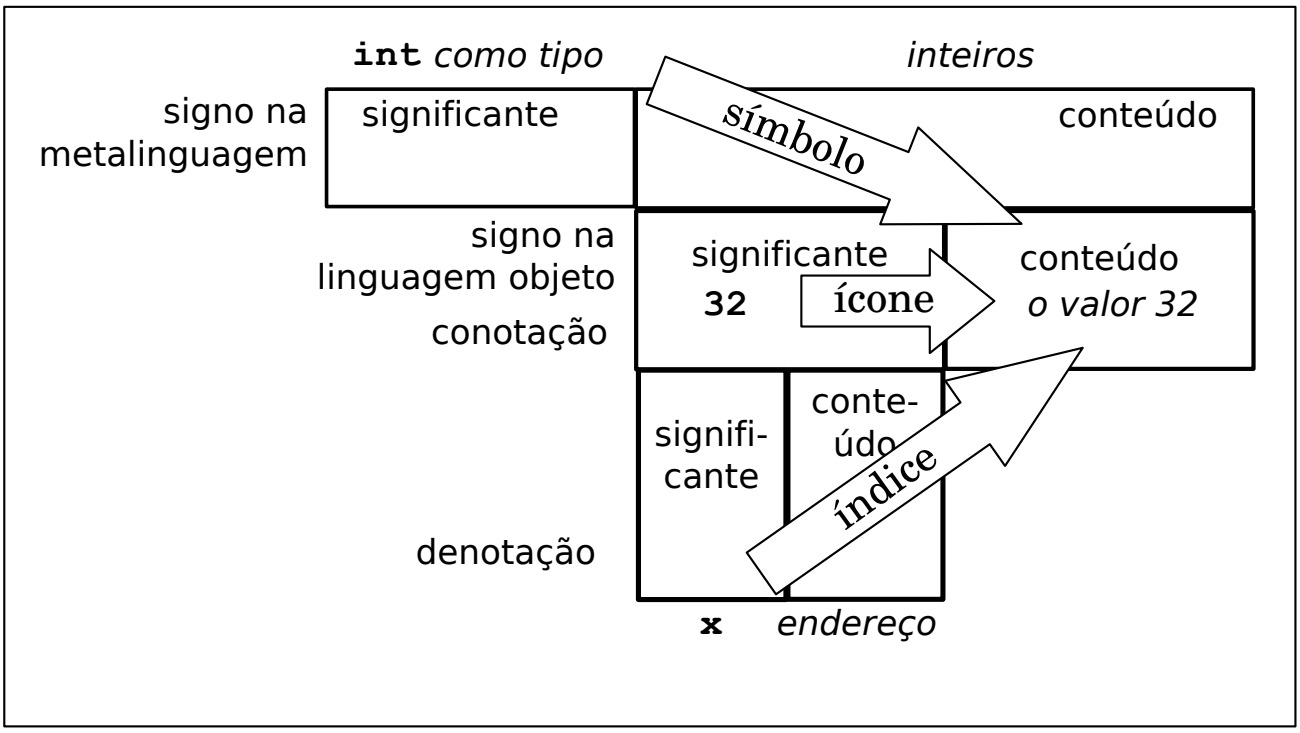
Ícone é "um signo que representa algo meramente porque lembra esse algo" (TANAKA-ISHII, 2010, p. 104; CP 3.362, 1885), "compartilhando caracteres do objeto" (p. 104; CP 4.531, 1905).

Índice é "fisicamente conectado ao seu objeto; formam um par orgânico, mas a mente interpretativa não tem nada a ver com essa conexão" (p. 104; CP 2.299, 1897); "é uma referência" (p. 105; CP 2.283, 1903). O uso dos signos A, B e C numa afirmação formal 'A e B são casados e C é seu filho' são índices (p. 105; CP 2.285, 1893).

Símbolo é um signo que "refere-se ao objeto que denota em virtude de lei, usualmente uma associação de ideias gerais" (p. 105; CP 2.240, 1903); "qualquer palavra comum como 'dar', 'pássaro', 'casamento', é um exemplo de um símbolo" (p. 105; CP 2.298, 1893).

A autora entende intuitivamente a correspondência do ícone e do índice com a primeiridade e secundidade, respectivamente, mas as razões pelas quais o símbolo corresponde à terceiridade depende da essência desta última, que discutirá no próximo capítulo de seu livro (TANAKA-ISHII, 2010, p. 105). Não obstante, consegue classificar os signos na expressão '`int x := 32`' da seguinte forma: '`32`' é um ícone pois denota imediatamente o valor, que é um conjunto de padrões de bits representando zeros e uns; '`x`' é um índice na medida em que refere-se ao endereço onde está localizado o valor, e '`int`' é um símbolo, pois um tipo de dado embute uma ideia geral a respeito do valor (p. 105). Desse modo, faz em seguida uma associação entre os dois modelos de signo conforme a Figura 10, que é similar à Figura 9, tendo sido as camadas da linguagem objeto e conotação da Figura 9 sido sobrepostas na Figura 10, "uma vez que indicam o mesmo signo" (p. 106)

Figura 10 – Correspondência das duas classificações de signo, traduzida e adaptada de Tanaka-Ishii (2010, p. 106)



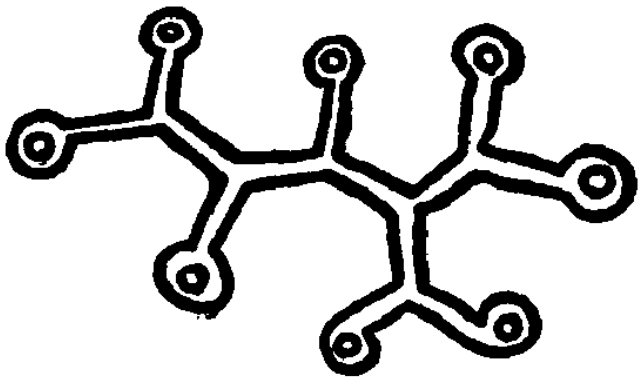
3.3.2 Capítulo 7 - Três tipos de conteúdo em programas

O capítulo 7 considera os vários tipos de conteúdo representado e examina como os signos estão envolvidos em tal representação (TANAKA-ISHII, 2010, p. 109). O texto inicia aprofundando as categorias peirceanas, que para ela são "um modo de classificar diferentes tipos de formas em termos do número de formas envolvido" (p. 111), tendo sido brevemente vistas no capítulo anterior de seu livro. Para ela, "a questão subjacente das categorias universais é procurar as formas básicas mínimas entre todas as formas. Nas categorias universais, a forma da primeiridade envolve apenas uma forma, a da secundidade envolve duas formas e a da terceiridade envolve três formas" (p. 111). E, partindo das asserções nas quais Peirce afirma que suas categorias são essencialmente diferentes e suficientes, não havendo necessidade de uma quarta categoria, faz a seguinte colocação:

> É abundantemente claro que primeiridade e secundidade são diferentes. Uma forma sem relacionamento é essencialmente diferente de duas formas relacionadas. É menos claro, entretanto, o que essencialmente distingue secundidade de terceiridade. Por que não decompor terceiridade em secundidade e primeiridade? Por que é possível dizer que três é necessário e suficiente? (TANAKA-ISHII, 2010, p. 111).

O livro apresenta a famosa figura de Peirce em (CP 1.371, 1885), reproduzida na Figura 11, como uma resposta "intuitiva" que o pensador dá à questão: a figura mostra que é

Figura 11 – Três elementos essenciais numa rede de estradas: terminadores, conexões e ramificações (CP 1.371, 1885)



essencial haver relacionamentos de três termos para relacionar qualquer número de formas livremente. Mas a autora considera que o argumento não representa suficientemente a natureza das formas nas categorias universais. De acordo com ela, se a primeiridade não pode ser referenciada por nenhuma outra forma, então:

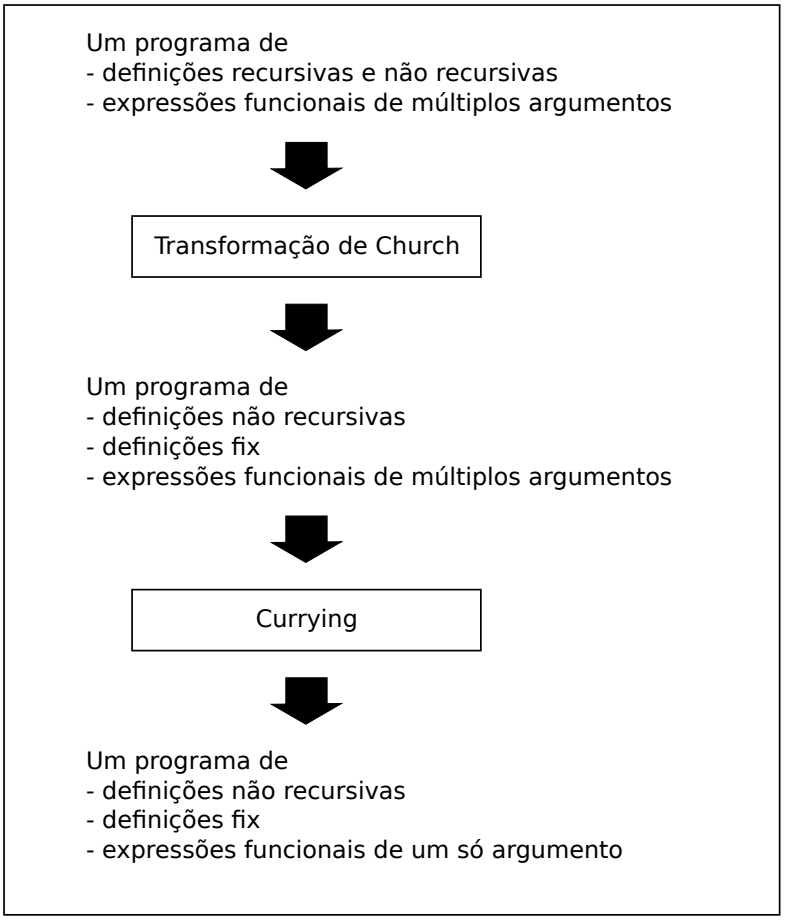
> [...] sua significação como forma é duvidosa, uma vez que não terá nenhuma relação com nenhuma outra forma. Portanto, deve ser o caso que as categorias universais estipulam formas do ponto de vista *funcional*; ou seja, tem de haver uma distinção entre uma forma *referir-se a* outra e uma forma *ser referida por* outra. Em outras palavras, a primeiridade por si não se refere a nenhuma outra forma, mas pode *ser referida por* outras formas. Se é assim, uma visão funcional similar tem de se aplicar a secundidade e terceiridade. Considerando secundidade, uma forma de secundidade pode ser referido por qualquer outra forma de secundidade, e também, formas que são referidas por secundidades podem ser referidas por quaisquer outras forma de secundidade. Em outras palavras, várias relações podem ser construídas a partir de apenas primeiridades e secundidades, através da conexão de formas de várias maneiras. Então, a validade da explanação de Peirce a partir do ponto de vista da conectividade das formas se torna questionável, e também é questionável se a terceiridade é necessária, em adição à secundidade e primeiridade (TANAKA-ISHII, 2010, p. 112).

Após fazer algumas considerações sobre outros autores que também apresentaram a ideia de que haveria somente três formas, a autora esclarece que essas questões das categorias universais serão consideradas dentro do paradigma funcional de programação porque ela considera que as categorias universais de Peirce têm uma natureza funcional.

O texto a seguir apresenta duas transformações possíveis em programas escritos em linguagens de paradigma funcional, que a autora chama de "*currying*" e "*transformação de*

Church", em cujos detalhes não entraremos aqui. Basta-nos saber que a primeira permite que uma aplicação funcional de múltiplos argumentos seja transformada em múltiplas aplicações funcionais de funções de um só argumento (TANAKA-ISHII, 2010, p. 115), e a segunda permite transformar uma definição auto-referente —e portanto recursiva— em um conjunto de definições não recursivas mais uma função de ponto fixo '`fix`', que segundo o texto também é recursiva. Dessa forma, ela demonstra, como mostrado na Figura 12, como se pode transformar qualquer programa escrito no paradigma funcional

Figura 12 – Esquema utilizado na transformação dos programas, traduzido e adaptado de Tanaka-Ishii (2010, p. 121)



em um outro programa contendo somente três tipos de elemento:

- funções não recursivas
- funções de ponto fixo '`fix`'
- expressões funcionais de um argumento

O fato de haver programas que não podem ser reduzidos a menos de três relatos determina, para a autora, a necessidade da terceiridade, sendo esta genuinamente representada pela

função auto-recursiva. Cada um dos três elementos é associado a uma das categorias de Peirce: os argumentos únicos das funções pertencem à categoria da primeiridade, as funções em si à secundidade e a função '`fix`', auto-recursiva, à categoria da terceiridade. Algumas considerações adicionais da autora:

> Terceiridade é um tipo de conteúdo. Auto-referência como terceiridade significa que algo é estipulado por meio de si mesmo em relação com algum outro conteúdo. Articulação de tal conteúdo requer um meio de se referir ao conteúdo. Além disso, esse meio tem de ser especulativo, uma vez que o alvo a ser estipulado será definido por meio de si mesmo. Em outras palavras, signos e um sistema de signos têm um papel fundamental na realização do conteúdo da terceiridade. Eles oferecem um meio de articular terceiridade [...] Além disso, uma vez que a descrição é obtida pelo uso de signos, ela se aplica a qualquer conteúdo que cumpra a descrição. Uma descrição usando signos neste sentido envolve uma abstração. Este modo de considerar terceiridade como um abstração explica por que Peirce considerava um signo como representativo da terceiridade [...] uma vez que a abstração consiste de reconsiderar conteúdo reflexivamente em comparação com conteúdo similar. Ademais, a razão por que o símbolo de Peirce é considerado como terceiridade a respeito da relação entre o signo e o conteúdo [que para a autora, lembremos, é o objeto imediato] deve estar claro agora (TANAKA-ISHII, 2010, p. 123).

Portanto, de acordo com o texto, as respostas a respeito da terceiridade procuradas no início do capítulo podem agora ser vistas, ao menos no caso da computação.

3.3.3 Capítulo 8 - Uma instância versus A instância

A autora inicia dizendo que a questão que quer considerar aqui é a individualidade das instâncias geradas por computador: daí a contraposição entre *uma* instância e *a* instância, que é uma "questão mais séria em computação dada a facilidade em obter reprodução perfeita". Nos informa que "nesse capítulo a hecceidade significa uma propriedade que *a* instância possui mas *uma* instância não" (TANAKA-ISHII, 2010, p. 127). Isso se associa à programação na medida em que "a meta do programa é descrita no nível de uma *classe* [num sentido mais geral que o termo técnico da programação orientada a objetos], através de um processo indutivo de modelagem do propósito da classe. O processamento real da informação é então conduzido através de um processo dedutivo utilizando objetos concretos chamadas *instâncias*, que são geradas através do processo de *instanciação*" (p. 127-128). Ou seja, está se preocupando aqui com o uso do produto final do programa, que em nossa nomenclatura é o software. Segundo ela, "descrição computacional sempre diz respeito à atividade humana de modelagem de um propósito através da abstração indutiva, geração de instancias através de instanciação, e cálculo através da dedução", e, dos três processos, a instanciação "foi esquecida na hecceidade dos escolásticos" (p. 129).

A questão permanece um tanto obscura até o ponto em que coloca que "se o usuário de um programa tem um contexto específico em mente e sabe exatamente como gerar instâncias apropriadas, então a instanciação não é um problema. Ademais, se a instanciação diz repeito apenas à seleção aleatória dentre um conjunto possível de instâncias pertencentes a uma classe, então o processo é trivial". Prossegue em seguida: "o tema deste último capítulo da Parte II é instanciação voltada para a obtenção d*a* instância, que é um outro problema que tem a ver com tipos de conteúdo". E esclarece: "a motivação dessa discussão repousa na dificuldade em obter *a* instância ao invés de *uma* instância dentro da computação" (p. 129). A importância disso para a autora fica clara quando expõe um estudo de caso de uma narrativa digital do qual participou, "um software chamado Mike, que gerava comentários automáticos em linguagem natural e em tempo real para jogos de futebol jogados por robôs autônomos e programas de computador numa série de competição chamada Robocup". De acordo com ela, a partir da teoria narrativa de estórias russas construída por Vladimir Propp, um programador "sente que poderia escrever um gerador automático de estórias russas". Não obstante, segundo ela, o problema nesses sistemas não está na modelagem: está na geração de uma instância narrativa significativa (p. 130). O programa Mike gerava narrativas em tempo real a partir de eventos disparados dentro do jogo, e a saída real era sorteada entre os narremas (unidades narrativas) possíveis; o palavreado do narrema era também sorteado, de modo que o resultado fosse único. Mas a impressão da audiência a respeito dos comentários de Mike era de que eram "repetitivos e mecânicos". Segundo ela, o principal motivo disto é a falta de uma metodologia bem concebida para instanciação (p. 130–131). Afirma ela que jogos de computador baseados em papéis (*role-playing computer games*) têm uma arquitetura semelhante ao sistema Mike no que diz respeito ao uso de narremas e sintaxe narrativa, mas os primeiros despertam uma reação oposta à reação ao sistema Mike, e a principal diferença reside no processo de instanciação utilizado em cada gênero (p. 132).

O texto prossegue fazendo considerações filosóficas a respeito da relação entre o problema dos universais e a instanciação; para ela a questão dos universais degradou a importância das instâncias. Segundo ela, é visível a perda de valor das instâncias no decurso de uma mudança nos métodos de instanciação: de (1) instância irreprodutível ou de reprodução limitada para (2) cópias de um objeto do mundo real, como fotografias ou performances musicais e daí para (3) instâncias reprodutíveis por si, como gráficos computadorizados. E a questão passa a ser: como restaurar hecceidade em instâncias reprodutíveis? (p. 132–134).

O livro apresenta dois meios para restauração da hecceidade na instanciação computacional: otimização e interação. A otimização passa pela busca exaustiva entre as instâncias de uma classe, e essa busca restaurará a instância ótima dentro de certos critérios. Não é trivial de duas maneiras: pela rotina de otimização e pelo enorme volume

de casos a examinar; uma abordagem que pode ser adotada, segundo ela, é a de espelhar a natureza. O problema com o sistema Mike parece ter sido justamente a má escolha (aleatória) das instâncias. A recuperação da hecceidade através da interação envolve a escolha de um ser humano. A escolha feita por um ser humano de uma entre muitas alternativas atribui especial significância à instância escolhida, e permite que a narrativa flua no caso de um jogo de computador, por exemplo (p. 134–137).

Finalmente a autora especula a respeito da relação entre hecceidade e reflexividade, mostrando a reflexividade presente nos dois métodos apresentados: "a maioria dos procedimentos de otimização pode de fato ser transformada num procedimento complexo para obter tal ponto [o ponto fixo das funções recursivas], e a solução pode ser obtida por recursão". E a interação também tem relação com a reflexividade: "uma vez que dois sistemas estão envolvidos, o usuário e o computador, ambos são influenciados pela interação. O usuário pode evolver via interação com o sistema, que muda o usuário no que este aprende com o sistema e modifica seu próprio comportamento, o que sedimenta a estratégia para interações subsequentes. Reciprocamente, o comportamento do sistema pode ser modificado incrementalmente [...]. Cada interação modifica o comportamento do sistema, o que constitui o comportamento para a próxima interação". Assim, um esquema comum de reflexividade parece repousar por trás dos procedimentos de instanciação que restauram hecceidade, embora a autora não queira dizer que a reflexividade é o único esquema subjacente a toda hecceidade: na opinião dela, reflexividade é um dos meios que transforma *uma* instância n*a* instância (TANAKA-ISHII, 2010, p. 138). Conclui o capítulo tecendo considerações a respeito do tipo d*a* instância, lembrando a relação entre os componentes do programa, apresentados no capítulo anterior do livro, e as categorias de Peirce e, embora não chegue a nenhuma consideração conclusiva, afirma que "*a instância* é o ponto onde a distinção clara entre classe e instância se dissolve. Hecceidade tinha sido considerada como um tipo de noção oposta à universalidade, mas ela é de fato a transformação de uma instância na instância, a desconstrução de forma e matéria" (p. 140).

3.4 Parte III. Sistemas de Signos

Do Capítulo 9 ao último (12), examina sistemas de signos. O capítulo 9 compara dois tipos de sistemas de signos que chama de estruturais e de construtivos, associando-os respectivamente às linguagens naturais e às linguagens de programação. Os capítulos 10 e 11 levam em consideração os sistemas de signos no mundo e em interações com outros sistemas de signos. O capítulo 10 tece considerações sobre o sistema de signos computacionais em sua relação com o mundo (tomando uma visão que chamou de pan-semiótica), e o capítulo 11 leva em conta a reflexividade dos sistema, vista como a chave

para que o sistema de signos seja capaz de evoluir, o que pode ocorrer por auto-aumento ou por aumento mútuo em múltiplos sistemas. O capítulo 12 contém as conclusões.

3.4.1 Capítulo 9 - Humanos estruturais versus computadores construtivos

Segundo a autora, a meta deste capítulo é discutir as diferenças estruturais entre sistemas de signos humanos e mecânicos, realçadas pelo modo como lidam com a reflexividade; lembra-nos também que um "sistema de signos consiste de uma relação entre signos e suas interpretações". Essa discussão está, também, no nível semântico da linguagem de programação (ver p. 49), o que para linguagens naturais quer dizer que "o nível interpretativo dos signos das linguagens naturais é examinado apenas no nível interno à linguagem, desconsiderando objetos do mundo real externos, e considerando os significados dos signos das linguagens naturais como sendo constituídos através de suas definições e usos na linguagem" (TANAKA-ISHII, 2010, p. 147).

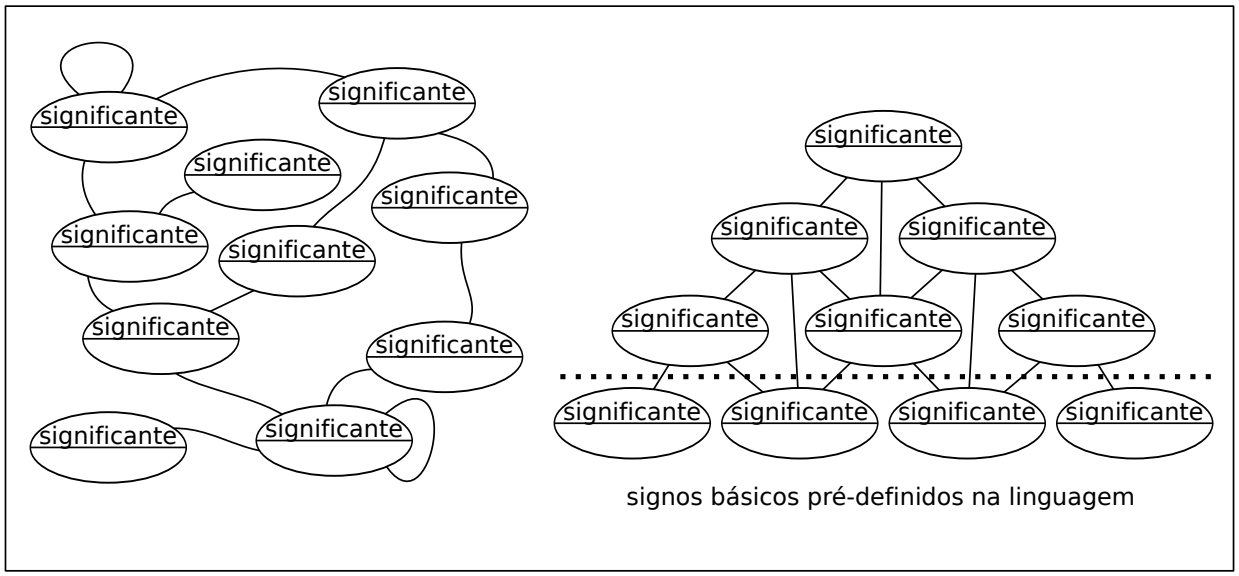
O texto inicia fazendo considerações a respeito da auto-referência nas linguagens: "alguma expressões linguísticas estão situadas nas margens da interpretabilidade; entre elas estão aquelas que exemplificam reflexividade" (p. 147). Prossegue com exemplos adicionais, citando inclusive algumas expressões que segundo a autora não têm problema para ser interpretadas, como por exemplo a função fatorial. Mas nos lembra que auto-referência introduzida no cálculo-λ estendido através do conceito de definição (exposto aqui na página 57) "é feita através do uso de um signo introduzido especulativamente que se refere a conteúdo que se tornará consolidado no futuro", o que "não pode evitar a introdução de conteúdo incerto —que pode ser mesmo vazio ou contraditório—no sistema". De acordo com ela, tais problemas são presentes tanto na linguagem computacional quanto na natural. A diferença entre ambas reside, segundo ela, na estratégia interpretativa para a reflexividade em cada um dos dois sistemas de linguagem (p. 147–149).

Apresenta inicialmente o sistema estrutural lembrando que, "dada uma expressão, humanos sempre têm a escolha de abandonar qualquer tentativa de interpretação se o processo de interpretação estagna". Segundo ela, tal estratégia permite a interpretação robusta de expressões problemáticas com auto-referências. E ao mesmo tempo "gera um sistema de signos no qual muito do conteúdo concreto dos signos fica ambíguo mas ainda existe dentro do sistema de linguagem". A autora afirma que este processo pode ser expresso por um "modelo gerativo": um signo é introduzido especulativamente, com seu conteúdo concreto deixado ambíguo, ambiguidade que vai sendo esclarecida graças a definições adicionais e uso. Adicionalmente, "os usos e conteúdo do signo mudam com o tempo, e o todo representado pelo significante evolve", e como resultado "é frequentemente difícil definir o conteúdo e significado concreto de um signo da linguagem natural". Afirma: "raramente o conteúdo exato do signo é explicitamente obtido, o que requereria resolver

auto-referências e obter um valor explícito, como no cálculo do fatorial". O significado de um signo da linguagem natural portante existe "flutuando na rede de signos que são usados em expressões que se referem ao signo. Em outras palavras, um signo é referido a partir de uma expressão que consiste de signos referidos a partir de muitas outras expressões, cujos signos são do mesmo modo referidos, e assim por diante, tal que o significado [*meaning*] do primeiro signo relaciona-se a todo o sistema. Um significante então representa tudo o que está relacionado ao signo com respeito ao conteúdo e usos. [...] É portanto o significante que articula o significado [*meaning*]; este *não* é nomeado pelo significante a posteriori". Segundo ela, a origem desta visão holística repousa no estruturalismo saussureano (TANAKA-ISHII, 2010, p. 149–151).

A seguir versa sobre o sistema construtivo: "computadores processam auto-referência de um modo totalmente diferente dos humanos", através de um processo de recursão, que pode não "convergir" e por isso levar a uma cadeia infinita de processamento. Isso coloca o computador em risco de cair num processamento infinito. Uma vez que não é possível criar uma computação capaz de julgar se outra computação[6] chega ao fim (de fato, esse é o tema do artigo de Turing citado neste trabalho (TURING, 1936)), a autora lembra que é responsabilidade do programador escrever um programa que chegue ao fim,

Figura 13 – Um sistema estrutural (esquerda) e um sistema construtivo, traduzido e adaptado de Tanaka-Ishii (2010, p. 155)



que conclua ("*halts*"). Para isso, manipulam cuidadosamente signos para garantir que os programas concluam. Isso "sublinha uma atitude fundamental entre programadores ao

[6] Falando de modo estrito, o texto se refere a computação como definida na página 31; sabemos que os equipamentos modernos são capazes de outros tipos de operação, embora não haja tratamento matemático adequado que permita lidar sem erro com essas outras operações, nem seja claro se esse conjunto adicional de operações dá ferramentas para resolver esse problema.

gerar programas: a de que estes deveriam ser construídos de um modo de baixo para cima ['*bottom-up*', em inglês]". Segundo ela, os programadores usam cuidadosamente signos que garantem que o processamento conclua, e esses signos são fornecidos primordialmente pelo sistema de programação. Esses sistemas incorporam técnicas que minimizam o risco de não conclusão, primeiro através de suporte teórico, e depois utilizando restrições linguísticas que dificultam a auto-referência, como a classificação hierárquica dos signos pelo seu *tipo* e a adoção da noção de escopo de um signo. Essas técnicas e componentes criam o que a autora chamou de sistema construtivo. Nele, elementos maiores são gerados como composições de elementos menores: "qualquer sistema de signos computacionais tem uma direção de construção, partindo de signos representando pequenas estruturas num nível mais baixo em direção a signos representando grandes estruturas num nível mais alto" (TANAKA-ISHII, 2010, p. 152–154).

Depois de apresentados os dois sistemas o texto ressalta as diferenças entre eles, ilustradas na Figura 13: "em analogia a Saussure [. . .], em um sistema estrutural não devemos começar com a palavra, ou termo, mas deveríamos começar do sistema, ao passo que em um sistema construtivo devemos começar com a palavra, ou termo, para construir o sistema" (TANAKA-ISHII, 2010, p. 154). Mais além:

> De certo modo, um sistema estrutural é formado naturalmente, sem requisitos formais, de modo que os signos se conectam uns com os outros arbitrária e livremente. O sistema resultante é holístico, irredutível a um núcleo mínimo. Uma vez que tal sistema faz o melhor dos signos reflexivos, o sistema de signos em si é reflexivo. De qualquer modo, um sistema construtivo é gerado a partir de um núcleo mínimo de signos que garantidamente concluem, e o sistema deve então ser construído de uma maneira de baixo para cima para preencher a requisição formal de concluir (TANAKA-ISHII, 2010, p. 154).

Daí tira algumas considerações importantes: segundo ela, "um aspecto da relação entre um sistema estrutural e um sistema construtivo é que o primeiro pode incluir totalmente o último, ao passo que o último pode incluir apenas parte do primeiro". Essa consideração, "derivada da diferença em como seres humanos e computadores interpretam auto-referências", é importante pois por causa dela um programador, ser humano que é, para escrever um programa é forçado a pensar de uma maneira construtiva em todos os signos do sistema: "negligenciar um signo pode levar a instantaneamente a comportamento desastroso". Para fazermos uma linguagem de computador mais amigável aos humanos, segundo ela, "um sistema inteiramente construtivo precisaria ser re-estruturado de modo a manipular signos formados estruturalmente". E a chave para isso "repousa no método de processamento da reflexividade, inclusive da auto-referência. Estudos relacionados a esta questão no domínio da ciência da computação tem sido feitos desde o nascimetno da da ciência da computação, como veremos no capitulo 11" (TANAKA-ISHII, 2010, p. 156).

3.4.2 Capítulo 10 - Signo e tempo

A autora nos informa que "os dois últimos capítulos do livro são dedicados a olhar para a natureza de um sistema de signos computacionais imersos em um ambiente", tomando uma visão "pan-semiótica". E começa, no presente capítulo, a focar na descrição da interação dentro do sistema de signos e conclui, no capítulo seguinte, considerando sistemas de signos comunicando-se com outros sistemas de signo. A questão subjacente ao presente capítulo é "como um sistema de signos se relaciona com o mundo exterior e o representa e como um signo é envolvido nesse processo" (TANAKA-ISHII, 2010, p. 158–159).

Inicia considerando que para habilitar a interação em um sistema computacional esta deve primeiro ser descrita dentro de um programa[7], e isso significa descrever eventos inesperados e imprevisíveis para o sistema. Tais eventos são descritos através de *mudanças nos valores dos signos*: um signo é alocado para interação, com valor indefinido, e um valor é mais tarde obtido de algum lugar fora do sistema e armazenado na posição de memória correspondente ao signo. A autora lembra que lidar com tal mudança de valor tem sido um problema há tempos. Uma solução seria a adoção da chamada *transparência referencial* (p. 160–161).

Para explicar o alcance dessa solução o texto inicia explicando a dificuldade relacionada à interação numa máquina de transição de estados; uma máquina de Turing, como visto em 2.2, é uma máquina de transição de estados. De acordo com a autora, um *estado* "é uma sequencia de zeros e uns dentro dos registradores da CPU, da memória principal, e da memória secundária (por exemplo, o disco rígido)". E quando a máquina executa uma operação através da CPU há uma mudança de estados que reflete o resultado da operação[8]. Portanto o estado do computador muda com o tempo. Nesse fato —a mudança do estado com o tempo— reside uma da grandes dificuldade para averiguar a correção dos programas: para sabermos se está correto é preciso ter certeza não só de que os comandos estão corretos, mas também que ocorrem na ordem correta. Nas palavras do texto: "quando um programa é suscetível ao erro, o programador tem de ler o programa para checar se a execução ocorre na ordem correta, verificando a correção do valor da cada signo. Um programa não pode ser verificado apenas pela checagem da correção de cada expressão". Isso equivale a executar o programa de um modo virtual para saber se está correto. (p. 161–162).

---

[7] Não é demais lembrar que não há modelos matemáticos satisfatórios para modelagem de tal interação: tal computação corresponderia á máquina de escolha de Turing, que não foi estudada por este, conforme comentado nas páginas 29 e 31.

[8] O conceito de estado é matematicamente equivalente ao de configuração completa numa máquina de Turing. A mudança de estado através da operação da CPU corresponde a um movimento da máquina de Turing (ver p. 30)

A transparência referencial (ver p. 41) é "uma restrição aplicada a um sistema de linguagem de programação que certifica que toda expressão tem um valor único". Sistemas de programação que usam uma linguagem baseada num paradigma funcional puro[9], como Haskell, introduzem a transparência referencial. Assim, o valor das variáveis não muda com o tempo, o que segundo a autora traz duas vantagens: primeira, "o resultado da execução não depende da ordem em que as expressões são avaliadas", e segunda, "melhora a reorganização dos signos e procedimentos". Introduz algumas dificuldades em termos de eficiência de execução, mas mesmo assim é uma restrição que "raramente é incluída na maioria das linguagens de programação largamente utilizadas porque a descrição de certos procedimentos fundamentais que são necessários para programas não é trivial sob a condição da transparência referencial" (p. 162–164).

Segundo o texto, para chegar a um sistema com transparência referencial é preciso reescrever procedimentos que não têm transparência referencial de um modo referencialmente transparente. Há o que a autora chama de *efeito colateral*, que são procedimentos que, por natureza, são fundados em mudanças de valores. Incluem "manipulação de exceções, não-determinismo, concorrência e, acima de todos, interação". O texto prossegue descrevendo técnicas de programação funcional que permitem implementar softwares (no sentido definido neste trabalho, na página 38) que permitem interação utilizando programas com transparência referencial. São as técnicas do *diálogo* e da *mônada*, em cujos detalhes não entraremos. Basta-nos dizer que, de acordo com a própria autora, essas soluções transformam o problema de ordenação temporal das mudanças em um problema de ordenação espacial (na memória do computador) dessas mudanças, o que não soluciona efetivamente o problema e eleva os custos de processamento a patamares proibitivos (p. 164–169).

Como consequência disso a autora propõe que "a interação deveria ser reconsiderada fundamentalmente em termos da natureza temporal dos signos". Considerando a semântica no nível de hardware e utilizando o modelo diádico de signo, segundo o texto, o significante "corresponde ao endereço de memória, ao passo que o conteúdo denota o valor contido no endereço". Do ponto de vista espaço-temporal, isso significa que o significante "carrega o aspecto espacial do signo, ao passo que o conteúdo captura o lado temporal do signo". Dentro dessa relação espaço-temporal, num computador, a espacialidade precede a temporalidade: "sem alocação de um signo, seu valor não pode ser alterado". Conteúdo sem significante não pode existir num computador, mesmo quando se tratam de termos-λ, cujo significante tem de existir ao menos no nível do hardware, segundo a autora. Portanto, no domínio da computação pode haver signos introduzidos especulativamente que significam conteúdo a ser estabelecido no futuro. E a questão é: qual é a significação de tal signo,

[9] A autora qualifica: "o termo *puro* significa um paradigma funcional sujeito à restrição da transparência referencial" (TANAKA-ISHII, 2010, p. 162).

sem conteúdo consolidado? (p. 169–170).

O capítulo prossegue lembrando que nos casos em que há interação, o signo alocado a ela passa de um valor inicial indeterminado para um outro valor que é desconhecido não porque ainda não foi calculado, mas porque veio do mundo exterior: o papel do signo na interação é "introduzir heterogeneidade de fora". E a consequência "é fundamental: a discussão até o momento parece sugerir o papel de um signo como meio transcendental de comunicação com o mundo externo, desconhecido e heterogêneo". Segundo o texto, esse pode ser o ponto de partido para o exame da natureza geral da interação num sistema de signos, tendo duas premissas que já foram vistas: a primeira, que "o signo possui uma natureza especulativa" e a segunda que "o sistema de signos tem de operar tendo o mundo externo como entorno", premissas que valem para sistemas de signos humanos também, de acordo com a autora. Depois de algumas considerações filosóficas envolvendo Heidegger, lembra-nos que a auto-referência, assim como o signo, também apresenta uma natureza especulativa (p. 170–173).

3.4.3 Capítulo 11 - Reflexividade e evolução

O último capítulo "considera a natureza reflexiva de uma sistema de signos novamente, mas desta vez do ponto de vista de todo o sistema [...]. Aqui, a noção de reflexividade é um caso especial de uma interação do sistema consigo mesmo via o mundo externo" (TANAKA-ISHII, 2010, p. 176).

Inicia afirmando que a linguagem natural é reflexiva em vários níveis. "No nível do ser humano, um sistema de linguagem evolve produzindo uma saída interpretável por si. O entendimento comum de que *afirmar/escrever é entender* mostra que a produção de uma expressão objetifica um pensamento como uma composição de signos, que reflexivamente se torna a entrada influenciando o pensamento, desse modo alimentando o entendimento". Segundo ela, através da reconsideração reflexiva da saída (*output*) auto-produzida, "um ser humano pode mudar, melhorar, e evoluir". Continua afirmando que "reflexividade ocorre entre múltiplos sistemas de linguagem natural, a saber, entre múltiplas pessoas" —o que nos indica que a autora considera que uma pessoa é um sistema de linguagem natural. Segundo ela, a reflexividade do sistemas de linguagem natural decorre da sua natureza estrutural (ver *Capítulo 9 - Humanos estruturais versus computadores construtivos*). E no caso das linguagens de computador a reflexividade no nível do sistema não é óbvia, em parte por sua natureza construtiva (idem), mas garante que os sistemas de linguagem computacional poderosos são todos reflexivos. Com isso especifica melhor o objetivo do capítulo, que é olhar a respeito da reflexividade de vários sistemas de linguagem computacional (p. 177–179).

Começa introduzindo o conceito de *homoiconicidade*, a característica dos sistemas de linguagens de programação que denota um programa de computador que tem o mesmo formato que os dados que são entrada e saída do programa. Um exemplo é a linguagem de programação Lisp. O termo homoiconicidade portanto indica se um computador pode produzir programa auto-interpretáveis. E a autora estende a significação do termo homoiconicidade para significar reflexividade, até para evitar confusão nos leitores de outros domínios que não a computação (p. 179–180). A respeito de definições a autora esclarece:

> O termo *sistema de signos* neste livro [...] significa uma relação entre signos e suas interpretações. Interpretação [...] significa processar uma expressão de acordo com as regras interpretativas pré-definidas do sistema de signos. Um *sistema de linguagem* é um tipo de sistema de signos consistindo de uma relação entre signos linguísticos e suas interpretações. Linguagens naturais e linguagens computacionais, ambos, consistem-se sistemas de linguagem. Alguns sistemas de signos, incluindo sistemas de linguagem, são produzidos pelo uso de um sistema de linguagem. No caso de sistemas de computador, a maioria é escrito em alguma linguagem de computador. Um *sistema de linguagem reflexivo* adquire alguma expressão como entrada e produz outra como saída, e partes da saída formam expressões interpretáveis pelo próprio sistema. [...] Linguagens naturais são sistemas de linguagem reflexivos [...]. A capacidade de um sistema de signos interpretar sua própria saída envolve *auto-aumentação*, ou seja, a capacidade do sistema mudar-se ou modificar-se, estender-se, melhorar, ou, possivelmente, evoluir (TANAKA-ISHII, 2010, p. 180).

O texto prossegue lembrando que a característica da aumentação não é limitada à auto-aumentação. Múltiplos sistemas podem se alimentar-se mutuamente, e essa alimentação é mais efetiva se esses sistemas são reflexivos. Afirma que quando mais de um sistema está envolvido, é importante saber se o sistema de linguagem é *aberto* ou *fechado*. É aberto quando o esquema de interpretação é público, ao passo que é fechado quando o esquema de interpretação mantém-se privado. Para sistemas abertos, múltiplos sistemas podem possuir exatamente o mesmo sistema interpretativo. Nesse caso, o resultado "obtido através do uso de múltiplos sistemas é o mesmo que o obtido por somente um sistema. [...] Em outra palavras, aumentação mútua entre sistemas abertos é qualitativamente equivalente a auto-aumentação". Em sistemas fechados os sistemas possuem diferente esquemas interpretativos, levando a influência adicional no resultado (p. 180–181). Sobre isso, afirma:

> Além disso, quando dois sistemas fechados comunicam-se utilizando linguagem, a influencia de uma expressão permanece indireta, uma vez que cada comunicador tem seu próprio intérprete fechado em si mesmo e o esquema de interpretação da contraparte é desconhecido. Um falante pode apenas gerar uma expressão assumindo como o ouvinte pode interpretar a expressão. Dentro desse assumir, a reflexividade tem um

> papel crucial na simulação do entendimento do outro pelo uso do próprio esquema interpretativo (TANAKA-ISHII, 2010, p. 182).

A autora conclui suas considerações a esse respeito afirmando que há controvérsia quanto aos "sistemas de linguagem naturais serem fechados ou não", e parte para o exame de como os sistemas computacionais de signos se encaixam quanto a reflexividade (p. 182).

Inicia propondo uma categorização das linguagens computacionais quanto à reflexividade: não reflexivas, finitamente reflexivas (caso das linguagens de pré-processamento) e infinitamente reflexivas. Prossegue introduzindo o conceito de compilador e interpretador: o interpretador é um sistema que "analisa diretamente as expressões de uma linguagem e as executa". Compilador é "software que traduz um programa em outra linguagem com um interpretador", lembrando que normalmente essa segunda linguagem é a linguagem de máquina definida para a CPU, e portanto o próprio equipamento é o interpretador. Portanto, "a criação de um sistema de linguagem fundamentalmente significa produzir ou um interpretador ou um compilador". Apresenta então a interessante questão da evolução da linguagem C: o compilador da linguagem foi escrito na própria linguagem. Em resumo, um pequeno subconjunto de C, suficiente para construir o compilador, foi construído na linguagem de montagem utilizada para controlar a CPU. Esse pequeno subconjunto foi utilizado para construir a versão-zero do compilador, uma versão-um foi compilada a partir desta, e assim por diante, até chegar ao estado atual de evolução. Segundo a autora, esse processo poderia ser automatizado, o que requereria duas funcionalidades adicionais: (1) o sistema de linguagem deveria ser capaz de produzir uma nova versão do programa auto-interpretável. Aqui, o ponto chave é o planejamento da atualização: com estender ou melhorar a linguagem, o que hoje em dia é feito por humanos. E (2) o sistema de linguagem precisaria uma moldura de trabalho para interpretar o programa modificado durante a interpretação do programa anterior, o que segundo a autora pode ser obtido através da meta-programação. Finalmente, a autora faz considerações a respeito da reflexividade de um sistema de sistemas de computadores, levando em consideração o fato de serem ou não abertos, e demonstrando a reflexividade presente nesse caso, que tem as mesmas limitação dos casos já vistos (p. 182–190).

Sua conclusão é de que os sistemas de signos computacionais são inerentemente reflexivos, o que é "a natureza de um sistema de signos em geral. Não se sabe, não obstante, como subordinar a reflexividade em uma direção específica nos sistemas computacionais, e mais importante, essa direção não é conhecida. Estipular tal direção começa por situar outros sistemas de signos dentro de um sistema de signos, [...] e também por situar o sistema de signos em si mesmo [...]. Nos sistemas de signos naturais, também, tais direções foram encontrados ao longo do curso de muitos anos, e a estipulação não é uma questão simples. A busca por tal direção foi e continuará sendo uma questão crítica na

evolução dos sistemas de signos" (p. 191).

3.4.4 Capítulo 12 - Conclusão

O texto começa recapitulando a motivação do livro e o caminho percorrido:

> O objetivo deste livro foi aplicar a teoria semiótica ao campo da programação de computadores e considerar a essa luz as propriedades gerais dos signos e sistemas de signos através do conceito de reflexividade. Programas de computador são descritos em linguagens formais bem definidas que têm processos interpretativos que são externos àqueles das linguagens humanas. Na escrita desse livro, as propriedades gerais dos signos e as características específicas dos signos computacionais foram estudadas em comparação com os signos das linguagens naturais. Certas partes da moldura semiótica foram reformuladas hipoteticamente através de sua aplicação a tais sistemas formais. Ao mesmo tempo, as significâncias das teorias de programação de computadores foram reconsideradas do ponto de vista humanístico, uma abordagem que trouxe luz a diversas razões pelas quais os programas de computador estão necessariamente no seu estado atual (TANAKA-ISHII, 2010, p. 193).

A autora prossegue descrevendo o que foi tratado em cada parte do livro e chega a suas considerações finais: "subjacente a este livro está a questão das diferenças entre homens e máquinas. Têm havido discussões substantivas a respeito d*o que computadores não podem fazer* [...]. [E]ste livro tentou considerar esta questão em termos da bancada comum dos sistemas de signos". Segundo ela, o mundo da computação consiste em última instância de signos, e "o pensamento humano pode ser considerado um sistema de signos, exatamente como pensadores como Saussure e Peirce argumentaram. [...] Vimos que não há diferença na natureza de um signo individual entre sistemas de signos humanos e mecânicos. Um signo se funda na sua introdução especulativa sem nenhuma garantia de que vá adquirir algum conteúdo final e concreto. [...] Um sistema formado de tais signos torna-se naturalmente suscetível à reflexividade, e a estratégia para lidar com a reflexividade determina que tipo de sistema de signos ele se torna. Ao longo deste livro, de fato, vimos que muitas dificuldades que não existem em sistemas humanos aparecem quando a reflexividade é processada em máquinas" (p. 195–196).

Prossegue lembrando que as formas computacionais foram desenvolvidos para evitar reflexividade ambígua, afirma que o potencial para explorar reflexividade permanece limitada para máquinas, e sistemas de computador estão longe de evoluir. Prossegue a autora: "uma vez que signos são reflexivos, a questão central das teorias computacionais tem sido reflexividade, e a resposta à questão d*o que os computadores não podem fazer* ainda repousa na reflexividade. Em outras palavras, uma limitação chave das máquinas ainda parece ser suas limitações em processar reflexividade" (p. 196). Conclui:

> Não obstante, sistemas de linguagem computacional têm sido potencialmente reflexivos desde sua invenção, do mesmo modo que sistemas de linguagem humanos. Por natureza, nós seres humanos soubemos como fazer o melhor desse caráter auto-referencial em nossos sistemas de linguagem, mas ainda não entendemos completamente como fazê-lo em nossos sistemas computacionais. Tentar preencher o espaço entre esses dois sistemas de signos, de qualquer modo, promete vantagens e desvantagens. Se os signos computacionais forem articulados como os da linguagem natural humana, esses signos tornar-se-iam mais robustos e sofreriam menos dos problemas causados pela reflexividade. Ao mesmo tempo, o esquema interpretativo tornar-se-ia fechado num sistema computacional, o que diminuiria a controlabilidade do sistema (TANAKA-ISHII, 2010, p. 196–197).

Enfim, segundo o trabalho, a história dos sistemas de linguagens de computador pode ser tomada como um esforço contínuo para descobrir novos meios de aplicar reflexividade para tornar tais sistemas mais dinâmicos e evolucionários. O livro pretendeu revisar esse esforço a partir do ponto de vista dos sistemas de signos para entender "de onde vimos, onde estamos, e para onde iremos a seguir" (p. 197).

4 Semiótica peirceana

Charles Sanders Peirce (1839-1914) apresentou uma maneira única e totalmente coerente de ver o mundo, inaugurando um modo de pensar. Sua obra é vasta, complexa, extremamente coerente e, de certo modo, desconhecida: parte dela permanece não publicada, estando somente os manuscritos originais disponíveis para leitura. Acreditamos que a aplicação de suas ideias a qualquer campo do conhecimento passa pela compreensão e aplicação desse modo de pensar. Portanto, e seguindo um caminho um pouco diferente do rumo traçado por inúmeros intérpretes de sua obra, passamos a seguir a explicar a diferença entre o modo peirceano de pensar e o modo mais comumente adotado hoje em dia.

4.1 Duas maneiras de pensar

Esta seção e grande parte deste capítulo foram escritos de maneira a se assemelhar ao modo mais comum de pensar hoje em dia; chamemos este modo de pensar de "cartesiano", em referência ao filósofo francês René Descartes, considerado o fundador da filosofia moderna com o *Discurso do Método* (DESCARTES, 2004, 1ª publicação em 1637). Escolhemos essa maneira de escrever com a finalidade de destacar a diferença entre o modo de pensar cartesiano e o modo de pensar peirceano. Embora a maneira peirceana seja mais completa —e isso se observa com a constatação de que a epistemologia peirceana explica a cartesiana, mas o contrário não ocorre—, é também menos familiar à maioria, o que a torna quase enigmática dependendo da natureza do assunto. Uma vez que a natureza do assunto do presente trabalho é simples se comparada às questões filosóficas fundamentais abordadas por Peirce, acreditamos que se ganha mais do que se perde adotando aqui esse modo de expor.

Fique claro antes que não se trata de escolher entre uma e outra forma de pensar, nem de abandonar o método cartesiano, que tem se mostrado tão eficiente em certos ramos do conhecimento. Ao contrário, trata-se de expandir as ferramentas do pensamento cartesiano agregando a elas métodos mais abrangentes, e tão rigorosamente lógicos quanto o pensamento de Descartes.

4.1.1 O método cartesiano

Descartes lançou um conjunto de quatro preceitos lógicos para um método para "chegar ao conhecimento de todas as coisas de que meu espírito fosse capaz" (DESCARTES,

2004, p. 54–55), enumerados a seguir para facilitar referência futura —Descartes mesmo não os numera:

1. Não aceitar nada como verdadeiro, apenas o que fosse indubitável.
2. Dividir cada dificuldade em em tantas parcelas possíveis e que fossem necessárias para melhor resolvê-las.
3. Conduzir por ordem os pensamentos, começando dos objetos mais simples e fáceis de conhecer, para subir aos poucos, até o conhecimento dos objetos mais compostos.
4. Fazer enumerações tão completas e revisões tão gerais que dessem a certeza de nada omitir.

Essencialmente estendeu os métodos do geômetras às demais áreas de conhecimento. A engenhosa aplicação do método, pelo próprio Descartes, para responder questões filosóficas fundamentais levou-o a conclusões difíceis de refutar, embora muito se tenha tentado (DESCARTES, 2004, p. 69–73):

- *A prova da sua própria existência* resumida no lema "penso, logo existo". A prova impressiona pois o autor inicia o argumento "duvidando" de tudo o que o leitor comum julga indubitável —seus sentidos, seu raciocínio, suas memórias— para em seguida argumentar que não podia duvidar que estava pensando; e se estava pensando, necessariamente existia.
- *A separação entre mente e matéria* pois sendo possível imaginar sua existência sem corpo, mas sendo impossível imaginar sua existência sem pensamento, conclui naturalmente que ele, Descartes, é, em essência, pensamento, que por se comportar de maneira tão diferente é uma substância completamente diferente da matéria.
- *Como reconhecer verdades* em decorrência da identificação de uma única verdade (sua própria existência): as verdades são "coisas que concebemos de maneira muito clara e distinta; há apenas alguma dificuldade em observar bem quais são aquelas que concebemos distintamente".
- *A prova da existência de Deus* que decorre da observação de sua própria imperfeição (já que duvidava) e a pressuposição da existência de algo que lhe apontasse para a perfeição, ou seja, Deus.

É importante observarmos que as noções de clareza e distinção, necessárias para reconhecer as verdades, são abordados em outra obra sua (DESCARTES, 2014, §XLV–XLVI, 1ª

publicação em 1644) da seguinte forma: "claro" é o que se pode enxergar através de uma impressão forte, em contraposição, por exemplo, a "obscuro", que não deixa uma impressão forte, mas fraca; "distinto" é aquilo que "é tão preciso e diferente de todos os outros objetos de modo a abranger em si somente o que é claro": ou seja, embora clareza seja pré-condição para a distinção, pode haver coisas claras que não são distintas. Descartes cita como exemplo a percepção da dor: "quando qualquer um sente dor intensa, o conhecimento que ele tem da dor é muito claro, mas não é sempre distinto; pois os homens usualmente confundem-na com o julgamento obscuro que eles formam a respeito de sua natureza, e pensam que há na parte que sofre algo similar à sensação de dor da qual eles têm consciência, dela somente".

Algumas características do modo de pensar cartesiano podem ser inferidas a partir do que vimos até aqui.

- O primeiro preceito lógico, que preconiza aceitar como verdadeiro somente o indubitável, impõe o desafio (identificado por Peirce, como veremos adiante) de reconhecer uma verdade indubitável: afinal, basta que o sujeito não duvide para ser verdade? Mas e se a ausência de dúvida for consequência de irreflexão? Essas ponderações, se levadas adiante como fez Peirce, podem levar à conclusão de que o método cartesiano tem falhas importantes, e disso trataremos oportunamente.

  No momento essa consideração nos leva a um questionamento: algumas proposições são mais difíceis de duvidar que outras? E a resposta é sim: as proposições baseadas em fenômenos naturais repetíveis tendem a ser mais difíceis de duvidar. Por essa razão, o método cartesiano encontra *sucesso nas ciências naturais*, as que são baseadas em observações de fenômenos naturais repetíveis: química, física, biologia etc. Já as ciências cujas premissas não têm essas característica tendem a ter menos vantagens no uso do método cartesiano, como são as ditas ciências humanas: história, sociologia, filosofia etc. Nelas as conclusões são tão dubitáveis quanto as premissas.

- O segundo preceito lógico preconiza que uma dificuldade seja dividida em parcelas menores. Quão menores? O primeiro preceito ajuda a responder: tão pequenas quanto seja necessário para identificar de forma indubitável o que é verdade e o que não é. A partir dessa fragmentação das dificuldades é que se começa a arrazoar sobre elas, partindo do mais simples até chegar ao mais complexo. Os raciocínios passam a se mostrar baseados em verdades indubitáveis simples, que funcionam como os axiomas da geometria, que se compõem para chegar a conclusões mais complexas.

  Essa *tendência à axiomatização* tem a desvantagem de impossibilitar o uso do método para avaliar proposições essencialmente complexas —ou seja, que não podem ser decompostas em proposição mais simples sem que fiquem descaracterizadas.

- Ao preconizar que os raciocínios complexos partam dos mais simples, iniciando no que é indubitável e construindo a partir daí um arrazoado, o método cartesiano permite a construção de grandes edifícios teóricos a partir de premissas simples e raciocínios dedutivos —que dispensam verificação experimental (ver 4.2.4). O caso expoente é a própria matemática, um corpo de conhecimentos tão vasto que não pode ser inteiramente conhecido durante uma vida e que é baseado em tão poucos axiomas que se contam nos dedos. Termina havendo uma *supervalorização do encadeamento lógico nos raciocínios*, em detrimento, às vezes, da plausibilidade das conclusões.

  Uma consequência é que pequenas variações no que se considera indubitável leva a enormes variações nas conclusões. Resulta comum que corpos de conhecimentos sobre o mesmo assunto sejam muito diferentes, como é o caso, por exemplo, das próprias teorias semióticas. Outra consequência é que essa capacidade termina por gerar corpos de conhecimento extensos que não têm pontos em comum com outras disciplinas, dificultando, quando não impossibilitando, posteriores abordagens multidisciplinares.

- Ao determinar que a distinção é condição necessária para o reconhecimento da verdade se está privilegiando argumentos que apresentam tal distinção. Mas esta distinção pode ser obtida través de um artifício estilístico: ao contrastarmos um conceito ao seu oposto destacamos mutuamente ambos os conceitos apresentados, da mesma forma que um ponto preto é melhor destacado sobre um fundo branco do que sobre um fundo acinzentado. O resultado é que os textos que apresentam uma *tendência à análise dualista*, no estilo "ou isso ou aquilo", tendem a parecer mais verdadeiros.

  Um exemplo é o texto desta seção 4.1: *Duas maneiras de pensar*, escrito propositalmente nesse estilo: aqui se colocam as duas epistemologias lado a lado, e, ignorando todas as nuances intermediárias, praticamente as mostra como alternativas mutuamente exclusivas, embora isso nunca seja afirmado. As razões pelas quais esse tipo de oposição introduz uma impressão de clareza podem ser explicadas por argumentos da epistemologia peirceana, como se verá na subseção 4.1.2: *O método peirceano*.

4.1.2 O método peirceano

Peirce foi o primeiro filósofo a refutar completamente o método de Descartes e apresentar uma alternativa consistente (SANTAELLA, 2004, p. 32). Trata-se de um edifício filosófico cujo alcance está longe do escopo do presente trabalho. Basta-nos dizer que sua semiótica, ao lado da estética e da ética, ocupa a importante posição de "teoria do pensamento auto-controlado, ou deliberado" (CP 1.191, 1903): trata-se da posição tradicionalmente ocupada pelo lógica. E para Peirce a lógica é a semiótica: "todo pensamento sendo performado por meio de signos, a lógica pode ser considerada a ciência

das leis gerais dos signos" (CP 1.191, 1903). Prossegue: "possui três ramos: 1, Gramática Especulativa, ou a teoria geral da natureza e significados dos signos, quer eles sejam ícones, índices ou símbolos; 2, Crítica [também Lógica Crítica em outros textos de Peirce, cf. SANTAELLA, 1999], que classifica argumentos e determina a validade e grau de força de cada tipo; 3, Metodêutica [também Retórica Especulativa em outros textos de Peirce, cf. SANTAELLA, 1999], que estuda os métodos que devem ser perseguidos na investigação, na exposição, e na aplicação da verdade. Cada divisão depende daquela que a precede" (CP 1.191, 1903). No presente trabalho estudaremos principalmente em detalhes a primeira divisão; a segunda e a terceira aparecem em menos detalhes respectivamente em 4.2.4 e em 4.2.5.

As ideias básicas de seu edifício filosófico foram apresentadas em 1867 no seu artigo "Sobre uma nova lista de categorias" (PEIRCE, 1868). Aí expõe pela primeira vez o que se tornaria a coluna dorsal de seu pensamento, embora só tenha voltado a essas ideias em 1902, quando as retoma à luz da sua fenomenologia (SANTAELLA, 2004, p. 30). Nos anos de 1868 e 1869 publica, no *The Journal of Speculative Philosophy*, o que ficou conhecido como "a série sobre a cognição" ("*cognition series*"): três artigos que, de acordo com Santaella (2004) dão nascimento à sua epistemologia anticartesiana. São eles: "Questões concernentes a certas faculdades reclamadas para o homem" (1868), "Algumas consequências das quatro incapacidades" (1869) e "Fundamentos para a validade das leis da lógica" (1869) (CP 5.213-357). Neles critica o que chamou de "espírito do cartesianismo", caracterizado a seguir (PEIRCE, 1995; SANTAELLA, 2004, p. 35–36):

- Inicia a filosofia com base na dúvida. Peirce critica fortemente essa postura, pois não constitui dúvida genuína. É, segundo Peirce, um auto-engano, um abandono formal de crenças que tornará o pensador insatisfeito até que as retome formalmente (CP 5.265, 1868).

- Determina a consciência individual como o último teste de certeza: aquilo de que eu estou claramente convencido, o que quer que seja, é verdade.

- Toda inferência é conduzida por um fio único que parte de premissas indubitáveis: de acordo com Peirce, a filosofia deveria "imitar as ciências mais bem sucedidas" e:

  - adotar premissas que possam ser submetidas a exame cuidadoso e
  - não construir raciocínios que formem uma cadeia tão forte quanto seu elo mais fraco, mas sim "um cabo cujas fibras podem ser muitíssimo finas, contanto que sejam suficientemente numerosas e estejam intimamente conectadas" (PEIRCE, 1995, p. 259 – CP 5.265, 1868).

- Recusa-se a explicar, por definição, certos fenômenos, como por exemplo por que certos raciocínios devem ser considerados indubitáveis.

As razões de Peirce são expostas nos três artigos que formam um corpo complexo e apresentam uma alternativa à epistemologia cartesiana: o pensamento através de signos, que seria difícil de compreender sem o desdobramento posterior de sua obra e sem o entendimento das suas categorias, que passamos a descrever agora.

As categorias de Peirce

Iniciamos com uma ressalva: os conceitos que serão expostos a seguir são complexos e não podem ser decompostos em conceitos mais simples. Por isso não podem ser descritos do modo cartesiano, como um conjunto de axiomas simples que se estruturam num raciocínio complexo. Além disso a filosofia de Peirce não é dualista como a cartesiana —mas não só a cartesiana, diga-se (ROBINSON, 2012). Ou seja, Peirce não concorda com o dualismo mente-matéria que Descartes afirma; isso tem consequências importantes, sobretudo quando o assunto é semiótica. De acordo com o ponto de vista de Peirce signos estão em tudo e não são uma classe de fenômeno ao lado de outros objetos não semióticos: o universo inteiro, inclusive o homem, é constituído de signos (cf CP 5.448, 1905, rodapé). Isso nos livra, entre outras coisas, de uma dualidade entre sujeito e objeto, da "crença de que o sujeito é algo isolado, envolto numa aura de autonomia, dono de pensamentos sob seu perfeito controle. O sujeito, ao contrário, é linguagem, de modo que ele terá todas as características que a linguagem lhe dá" (SANTAELLA, 1998, p. 95). Além disso, Peirce se dizia adepto do "sinequismo": termo que criou e, em uma palavra, é a doutrina de que tudo que existe é contínuo (cf. CP 1.172, 1897). Uma consequência disso é que em seus textos são incomuns asserções sem ressalvas. Por exemplo, uma de suas definições de signo que veremos adiante: "Um signo ou representamen é tudo aquilo que, *sob um certo aspecto ou medida*, está para alguém em lugar de algo" (CP 2.228, 1897, ênfase nossa). Desse modo, Peirce abre mão do recurso estilístico da distinção por oposição que, como vimos (ver p. 84), parece ser uma característica de textos da linha cartesiana. Nossa sugestão é que o leitor tente construir o raciocínio de acordo com a sugestão de Peirce: não como uma corrente encadeada, mas como um cabo de muitas fibras que se entrelaçam.

Já no artigo de 1867 —"Sobre uma nova lista de categorias" (PEIRCE, 1868, CP 1.545-59)— as categorias estão expostas de um modo muito próximo ao que se tornariam depois, embora seu alcance não tenha sido percebido por Peirce. No artigo Peirce tenta desvendar como se pode a partir da multiplicidade do mundo chegar a (alguma) proposição que descreva algo no mundo. Identifica aí a necessidade das três categorias; sua aplicação ao que se tornou o edifício filosófico de Peirce só ocorreria décadas mais tarde. Mas antes: o que são categorias? Uma resposta de Peirce:

> Tento uma análise do mundo. Aquilo com que estamos lidando não é metafísica: é lógica, apenas. Portanto, não perguntamos o que realmente existe, apenas o que aparece a cada um de nós em todos os momentos de nossas vidas. Analiso a experiência, que é a resultante cognitiva de nossas vidas passadas, e nela encontro três elementos. Denomino-os *Categorias* (PEIRCE, 1995, p. 22, CP 2.84, 1902).

Portanto, categorias é o que se encontra na análise da experiência: categorias universais são comuns a todas as experiências. As categorias que Peirce encontrou foram nomeadas por ele de modo a não remeterem a nenhum conceito anterior, o que poderia gerar falsas associações. Inventou três palavras para elas: "Primeiridade" ("*Firstness*"), "Secundidade" ("*Secondness*") e "Terceiridade" ("*Thirdness*") (CP 4.3, 1898). Como o próprio Peirce em uma "Sinopse parcial de uma proposta para um trabalho sobre lógica" (CP 2.79-118, 1902), comecemos pela que parece mais fácil para nossas mentes cartesianas.

Secundidade

Para caracterizar a secundidade Peirce usa palavras como "força bruta", "binariedade" (CP 2.84, 1902), "obsistência" (CP 2.89, 1902), "conflito" ("*struggle*") (CP 1.322, 1903, CP 5.45, 1903). Sua característica é a de ser uma relação dual (identificam-se dois elementos) interagindo de modo inescapável. A forma mais fácil de observar a secundidade é olhar em volta: o leitor com certeza se vê cercado de objetos que mais ou menos permanecem de modo independente de seu pensamento. Creio que é a experiência mais simples a que alguém pode se submeter. Vemos o mundo, ele está lá; fechamos os olhos e ao abri-los, o mundo permanece lá. A secundidade neste fenômeno está na recusa dos objetos em se alterar, na insistência em permanecer. Em outro exemplo Peirce cita a "força bruta" dos fatos passados:

> Qual é, então, o fato que se apresenta a você? Pergunte a si mesmo: é o passado. Um fato é um *fait accompli*; o seu *esse* está no *praeterito*. O passado compele o presente, em alguma medida, no mínimo. Se você se queixar ao Passado de que ele é errado e não razoável, ele se rirá. Ele não dá a mínima importância à Razão. Sua força é a força bruta. Desta forma, você é compelido, brutalmente compelido, a admitir que, no mundo da experiência, há um elemento que é a força bruta (PEIRCE, 1995, p. 23, CP 2.84, 1902)

Mas é preciso resistir à tentação de aplicar a secundidade somente aos objetos materiais. A resistência à mudança aparece nos fenômenos mentais. O leitor por favor imagine agora, com o maior número possível de detalhes, a imagem de um dragão. Feito isso, responda a si mesmo qual a cor do dragão que imaginou. Não importa a resposta: a cor, e a resposta, permanecem, tanto quanto este texto que ora lê. Claro que é possível

imaginar outro dragão, ou o "mesmo" dragão com outra cor, ou ainda dizer que a escolha da cor no momento de imaginar o dragão não apresentou resistência, mas nada poderá mudar a cor do primeiro dragão imaginado: ela, junto com todo o dragão, também resiste —tanto quanto a memória permite, nesse caso particular. Ainda que se diga que essa persistência depende do alcance da memória do leitor, isso não importa: a secundidade está na resistência, e não na permanência eterna dessa resistência.

Peirce nos dá outros exemplos. No seu artigo sobre as máquinas lógicas de seu tempo (PEIRCE, 1887), após lembrar que as relações entre as partes do mecanismo dessas máquinas reproduzem as relações existentes no cerne do raciocínio lógico, Peirce termina afirmando que nossas mentes fazem substancialmente o mesmo quando constroem um diagrama de um raciocínio para em seguida observar os resultados. Isso mostra claramente que os efeitos de uma secundidade mental são os mesmos se uma secundidade física, e ajuda a ilustrar a essência dessa categoria. De fato, uma forma importante de secundidade mental está presente nos raciocínios dedutivos: deles Peirce afirma que as relações entre as conclusões e as premissas são tais que sua principal natureza é a secundidade (CP 2.96, 1902); voltaremos a isso em 4.2.4.

Convém observar a relação entre "força bruta" e "binariedade": se tomarmos como exemplo marido e esposa, uma simples dualidade, vemos aí também uma reação, uma vez que, lembra Peirce, é o marido que torna a esposa o que é e a esposa que torna o marido o que é. Essa força que se expressa na dualidade pode ser usada como artifício retórico. É ela que nos ofusca e dá às vezes a impressão de clareza, sendo utilizado como recurso estilístico de argumentação (ver p. 84 na subseção 4.1.1: *O método cartesiano*).

Primeiridade

Se a secundidade é exemplificada pela imutabilidade determinante dos fatos passados, Peirce associa a primeiridade à intangibilidade do presente absoluto:

> Consideremos agora o que poderia surgir como existindo no instante presente se estivesse completamente separado do passado e no futuro. Só podemos adivinhar, pois nada é mais oculto do que o presente absoluto. Claramente, não poderia haver ação alguma; e sem a possibilidade de ação, falar em binariedade seria proferir palavras sem significado (CP 2.85, 1902).

Não havendo a binariedade, "isso" de que Peirce trata não pode ser pensado diretamente. Mas algumas de suas características podem ser inferidas, como prossegue no mesmo parágrafo:

> Poderia haver uma espécie de consciência, ou ato de sentir, sem nenhum 'eu'; e este sentir poderia ter seu tom próprio. Não obstante o que disse William James, não creio que poderia haver uma continuidade como o espaço, [...]; e sem continuidade, as partes desse ato de sentir não poderiam se sintetizadas e, portanto, não haveria partes reconhecíveis. Não poderia nem mesmo haver um grau de nitidez desse sentir, pois tal grau é o montante comparativo de distúrbio da consciência geral sem sentimento (CP 2.85, 1902).

Trata-se de algo cujas características podem ser inferidas, mas que escapa a qualquer tentativa de racionalização. E é exatamente a isso a que Peirce se refere:

> De qualquer forma, essa será a nossa hipótese, e não tem nenhuma importância que ela seja ou não psicologicamente verdadeira. O mundo seria reduzido a uma *qualidade de sentimento não analisado* [ênfase nossa]. Haveria, aqui, uma total ausência de binariedade. Não posso chamá-la de unidade, pois mesmo a unidade supõe a pluralidade. Posso denominar sua forma de Primeiridade, Oriência [*Orience*] ou Originalidade. Seria algo que é aquilo que é sem referência a qualquer outra coisa dentro dele, ou fora dele, independentemente de toda força e de toda razão (CP 2.85, 1902).

Cumpre não confundir primeiridade com percepção. Santaella (1998) nos chama a atenção para o fato de que a percepção consciente é um fenômeno em que se destaca principalmente a secundidade por conta da insistência das coisas percebidas e a necessidade da ação física para tirar de nosso alcance perceptivo qualquer objeto (CP 7.620, 1903). Entretanto, é também um fenômeno em que a primeiridade tem grande importância. De fato, a percepção tem a característica de se fundar numa originalidade, num inesperado, numa primeiridade (CP 7.625, 7.630, 1903) que permanece e, por permanecer, pode então ser alvo de um juízo perceptivo[1] (SANTAELLA, 1998). No mesmo texto Santaella nos dá uma amostra que "assemelha-se a algo mais ou menos parecido" com a qualidade de sentimento de que nos fala Peirce, transcrevendo um exemplo dado num curso sobre percepção:

> Aqui estamos nós, nesta sala, no ato de estar —eu falando, vocês ouvindo. Existe um sentimento de qualidade do qual vocês só se darão conta porque estou falando dele, mas que nos acompanha o tempo todo, como uma melodia silenciosa ou um certo gosto na boca: é uma certa qualidade da temperatura da sala, certa luminosidade peculiar do entardecer, do som da minha voz, das sensações epidérmicas da roupa no corpo... Trata-se de um compósito de qualidades vagamente unidas num sentimento *in totum*, imediato, um mero *feeling*, impressão mais ou menos indefinida (SANTAELLA, 1998, p. 92).

É importante notar que ao falar na qualidade de sentimento, Santaella a traz para o foco da atenção, onde é percebida ("... vocês só se darão conta *porque estou falando dele*,

[1] Juízo esse que apresenta a categoria da terceiridade, que veremos mais adiante a partir da página 91.

. . . "). No momento da percepção, já não é mais uma qualidade de sentimento, próxima da primeiridade: é um fenômeno cuja secundidade é evidente. Essa é também uma característica daquilo que é meramente possível: enquanto possibilidade, ou potencialidade, é um fenômeno principalmente de primeiridade, até que eventualmente se realize, quando então é secundidade.

Peirce, num texto de 1887-1888 intitulado *A guess at the riddle* (*Um palpite sobre o enigma*) (CP 1.354-416)[2] caracteriza assim a primeiridade:

> A ideia do absolutamente primeiro deve estar separada de toda concepção de ou referência a algo outro; pois o que envolve um segundo é ele próprio segundo daquele segundo. O primeiro deve ser portanto presente e imediato, de forma a não ser segundo de uma representação. Tem de ser fresco e novo, pois se velho ele é segundo de seu estado anterior. Deve ser iniciativo, original, espontâneo, e livre; caso contrário é segundo de uma causa que o determina. É também algo vívido e consciente; só assim evita ser o objeto de alguma sensação. Precede toda síntese e toda diferenciação; não tem unidade nem partes. Não pode ser pensado articuladamente: afirme-o, e terá perdido sua inocência característica; pois asserção sempre implica a negação de algo outro. Pare de pensar nele, e ele terá ido! O que o mundo era para Adão no dia em que abriu os olhos, antes que traçasse quaisquer distinções, ou tenha se tornado consciente de sua própria existência —isso é primeiro, presente, imediato, fresco, novo, iniciativo, original, espontâneo, livre, vívido, consciente, e evanescente. Apenas, lembre-se de que toda descrição dele o falseia (CP 1.357, 1887-1888).

E no parágrafo seguinte mostra-nos também como a secundidade se articula com a primeiridade: "Mas não precisamos, e não devemos, banir do segundo a ideia do primeiro; ao contrário, o segundo é precisamente aquilo que não pode ser sem o primeiro." (CP 1.358, 1887-1888). A secundidade portanto não prescinde da primeiridade; e fenômeno semelhante ocorre com a terceiridade, como veremos mais adiante.

A primeiridade é também a categoria mais presente no fenômeno mental responsável pelo surgimento de ideias novas: o raciocínio abdutivo. Nele, as conclusões apresentam certo tipo de similaridade com o contido nas premissas, sem que necessariamente decorram delas (CP 2.96, 1902). Sua adoção provisória como hipótese permite que se desenvolva um raciocínio que a confirme ou não através da experimentação, conforme será detalhado 4.2.4.

[2] Esse texto continha indicações a respeito de material já escrito que deveria ser incorporado à versão final. Os editores dos *Collected Papers* decidiram fazer essa inclusão, numa tentativa de mostrar o texto como supõem que Peirce teria desejado. Em particular, os parágrafos CP 1.369-378, que corresponderiam ao capítulo 2, *The triad in reasoning* (*A tríade no raciocínio*), fazem parte dessa inclusão posterior.

No texto de 1887-1888 ("Um palpite sobre o enigma"), quando o alcance de suas categorias ainda não atingira a maturidade de anos posteriores, o próprio Peirce chama-nos a atenção quanto à aparente suficiência das duas categorias expostas até aqui: "Primeiro e segundo, agente e paciente, sim e não, são categorias que grosso modo nos habilitam a descrever os fatos da experiência, e elas satisfazem a mente por bastante tempo. Mas no fim elas se revelam inadequadas, e o terceiro é a concepção que então vem. O terceiro é o que constrói uma ponte sobre o abismo entre o primeiro e último absolutos, e os relaciona." (CP 1.359, 1890). Mais tarde, quando o papel da terceiridade no signo e na semiótica ficasse claro, é que se poderia perceber que as categorias da primeiridade e secundidade *não* nos habilitam a descrever os fatos da experiência: qualquer descrição tem como base a terceiridade, que é também essencial para a ação da mente. Vamos, então, ao terceiro.

Terceiridade

A aparente suficiência das duas categorias anteriores torna difícil a apreensão da terceira categoria peirceana. Peirce mesmo nos diz: "As ideias nas quais a Terceiridade é predominante são, como era de se esperar, mais complicadas, e normalmente requerem cuidadosa análise para serem apreendidas" (CP 1.338, sem data). Comecemos com o exemplo que é dado no texto abaixo, que complementa a visão do passado como secundidade e do presente absoluto como primeiridade:

> Consideremos agora o ser *in futuro*. Tal como nos outros casos, isto é meramente uma avenida que leva a uma apreensão mais pura do elemento que ela contém. Uma concepção absolutamente pura de uma Categoria está fora de questão. O ser *in futuro* aparece em formas mentais, intenções e expectativas. A memória fornece-nos um conhecimento do passado através de uma espécie de força bruta, uma ação bem binária, sem nenhum raciocinar. Mas, todo o nosso conhecimento do futuro é obtido através de alguma outra coisa. Dizer que o futuro não influencia o presente constitui doutrina insustentável. Equivale a dizer que não existem causas finais, ou fins. O mundo orgânico está cheio de refutações dessa posição. Uma tal ação (por causação final) constitui a evolução. Mas é verdade que o futuro não influencia o presente do modo direto, dualístico pelo qual o passado influencia o presente. Requer-se um instrumental, um meio. Todavia, qual pode ser esse instrumental, de que tipo? Pode o futuro afetar o passado através de um instrumental qualquer que, novamente, não envolve alguma ação do futuro sobre o passado? Todo nosso conhecimento das leis da natureza é análogo ao conhecimento do futuro, na medida em que não há nenhum modo direto pelo qual as leis tornam-se por nós conhecidas (PEIRCE, 1995, p. 25 CP 2.86, 1902).

Peirce aponta aqui para o cerne da questão: se primeiridade e secundidade fossem suficientes, como então surgiria a noção de futuro, uma vez que este não afeta o presente nem através da secundidade, nem através da primeiridade? Se houvessem somente essas

duas categorias, o futuro não nos afetaria, mas ele nos afeta: indiretamente, é claro. Afeta-nos, por exemplo, na medida em que nossas ações são também determinadas pelas nossas expectativas quanto a ele, como nos lembra Peirce: "cinco minutos de nossa vida acordada não se passam sem que façamos algum tipo de predição; e na maioria dos casos essas predições são realizadas" (CP 1.26, 1903). Há uma explicação para essa ação indireta. Peirce nos informa qual sua natureza: "O primeiro é aquilo cujo ser está simplesmente em si mesmo, não se referindo a nada nem se escondendo atrás de nada. O segundo é aquilo que é por força de algo para o qual é segundo. O terceiro é aquilo que é graças a coisas entre as quais ele medeia e que relaciona uma com a outra." (CP 1.356, 1890). Aplicando a ideia de mediação ao conceito de futuro temos uma pista melhor do que seja a ação da terceiridade: ela nos traz algo que não está aqui, e o faz de um modo não dependente nem da secundidade nem da primeiridade. Mas claro que essa ação se manifesta através de uma secundidade, mas uma secundidade diferente, uma secundidade que é uma degeneração de uma terceiridade. Quando escrevo qualquer palavra, a palavra "sol", por exemplo, esse conjunto de letras impresso na página é uma manifestação da secundidade, principalmente. Mas não é essa manifestação da secundidade que importa: é o que ela me traz. E isso que ela me traz é proporcionado pela terceiridade. Esse novo conceito ilustra o que Peirce quis dizer na última frase, quando afirma que não há nenhum modo direto pelo qual as leis da natureza tornam-se conhecidas. De fato, o que conhecemos diretamente são os efeitos dessas leis. Inferir quais leis são essas a partir de seus efeitos diretos exige essa nova categoria da terceiridade. Nela está a base do raciocínio. Outros trechos em que Peirce fala sobre a terceiridade (ou terceiro):

> Por terceiro, quero dizer o meio [*medium*] ou vínculo conectivo entre o primeiro e último absolutos. O início é primeiro, o final segundo, o meio [*middle*] terceiro. O fim é segundo, o meio [*means*] terceiro. O fio da vida é um terceiro, o destino que o corta, seu segundo. Uma encruzilhada na estrada é um terceiro, supõe três caminhos; uma estrada reta, considerada meramente como a conexão entre dois lugares é segundo, mas uma vez que implique na passagem por lugares intermediários é terceiro. [...] Lei como uma força ativa é segundo, mas ordem e legislação são terceiros. Simpatia, carne e osso, aquilo pelo que eu sinto os sentimentos do meu vizinho, é terceiro (CP 1.337, 1875).

> Algumas ideias de proeminente Terceiridade que, graças à sua grande importância na filosofia e na ciência, requerem estudo são generalidade, continuidade, difusão, crescimento, e inteligência (CP 1.340, 1895).

Particularmente esclarecedor é o trecho em que diz que a "mais fácil dessas [ideias onde a Terceiridade é predominante] que são de importância filosófica é a ideia de um signo, ou representação. Um signo representa [*stands for*] algo para a ideia que ele produz, ou

modifica" (CP 1.339, sem data). Isso nos introduz à semiótica de Peirce, o que faremos a seguir.

4.2 A semiótica de Peirce

A semiótica peirceana é mais que uma teoria dos signos: é uma teoria sígnica do conhecimento. O interesse de Peirce era em como se dá a cognição (SANTAELLA, 2012, p. 75). Tal escopo, embora seja, em nossa opinião, o mais logicamente adequado para um teoria de signos, termina gerando confusão quando se leva em conta outras teorias que também são intituladas "semiótica" ou "semiologia" e que estudam signos num sentido mais restrito ou mais focado na linguística, como é o caso, por exemplo, da semiótica de Saussure. Essa confusão de escopos contribui para que não exista um entendimento comum do que venha a ser semiótica, como apontou Nadin (2011, p. 95). Esse autor ressalta que "a aceitação de que a semiótica trata de signos [...] também não ajuda, pois não há consenso sobre o que é essa entidade chamada signo" (NADIN, 2011, p. 95). Mas o que nos interessa no momento é que a teoria de signos de Peirce suscita questões cognitivas que restariam incompreendidas sem a compreensão de suas categorias; tendo-as exposto, passamos à sua semiótica propriamente dita.

4.2.1 Signo

O conceito peirceano de signo não surgiu pronto. Sua evolução pode ser acompanhada numa leitura cronológica das definições de signo de Peirce, embora muitas vezes ele tenha recorrido a simplificações para se fazer entender, como confessou em 1908 numa carta a Lady Victoria Welby (PEIRCE,). Segundo Nöth e Santaella (2014), a terminologia de Peirce era idiossincrática, e pode-se encontrar referências ao signo inclusive numa fase "pré-terminológica". Embora o termo "signo" remeta claramente ao "algo que significa", trazendo à mente imediatamente a ideia de "palavra", convém lembrar que a sua natureza, como citado acima, é a da terceiridade. Então, embora de certa forma esteja correto dizer que o signo representa algo, o termo "representa" remete a um contexto que, em nossa opinião, tende a reduzir a compreensão do fenômeno. Melhor é dizer que, num processo chamado *semiose*, o signo relaciona três coisas:

> Um *Signo*, ou *Representamen*, é um Primeiro que está para um Segundo, chamado seu *Objeto*, em tal genuína relação triádica que é capaz de determinar que um Terceiro, chamado seu *Interpretante*, assuma a mesma relação triádica com seu Objeto na qual ele [o Primeiro] está para o mesmo Objeto. A relação triádica é *genuína*, ou seja seus membros são unidos por ela de uma maneira que não consiste em nenhum complexo de relações diádicas (CP 2.274, 1903).

A imagem de uma estrela de três pontas ou a letra "Y" invertida, com cada componente do signo em uma extremidade, embora seja muito utilizada para ilustrar o signo, *não* constitui uma relação triádica genuína: a estrela de três pontas traz juntas suas extremidades por ação da secundidade e não da terceiridade. Ou seja, não é o caso de que o signo por si traz seu objeto e seu interpretante de maneira irrefreável, embora possa haver casos em que algo semelhante a isso ocorra.

Entendemos que há uma enorme vantagem em não delimitar a ação do signo ao processo de representação somente. Pois isso permite analisar inúmeros fenômenos que não são estritamente de representação mas que não obstante são de terceiridade: um conjunto de traços no papel com o formato aproximado das letras "S", "O" e "L" relacionam-nos com a palavra "sol" —um fenômeno de representação e de terceiridade—, de um modo semelhante ao que a queda de um lápis relaciona-nos com a lei da gravidade —um fenômeno de terceiridade, mas não estritamente de representação. Uma outra definição, mais próxima do conceito usual de signo nos é traduzida por Nöth e Santaella (2014):

> Um signo ou representamen é tudo aquilo que, sob um certo aspecto ou medida, está para alguém em lugar de algo. Dirige-se a alguém, isto é, cria na mente dessa pessoa um signo equivalente ou talvez um signo mais desenvolvido. Chamo este signo que ele cria o interpretante do primeiro signo. O signo está no lugar de algo, seu objeto. Está no lugar desse objeto, porém, não em todos os seus aspectos, mas apenas com referência a uma espécie de ideia (PEIRCE, 1931-1958 apud NÖTH; SANTAELLA, 2014, CP 2.228, 1897).

Nöth e Santaella esclarece: "o signo não é uma classe de objetos, mas a função de um objeto no processo de semiose". E vai além, afirmando que o objeto de estudo da semiótica peirceana "não é bem o signo, mas a semiose", apoiando-se numa das definições de Peirce: "semiótica é a doutrina da natureza essencial e variedades fundamentais de semiose possível" (PEIRCE, 1931-1958 apud NÖTH; SANTAELLA, 2014, CP 5.488, 1907). No mesmo texto Nöth e Santaella nos traduz outra definição, em que o signo é definido em função da semiose: "semeiosis significa a ação de quase qualquer signo, e a minha definição dá o nome de signo a qualquer coisa que assim age" (PEIRCE, 1931-1958 apud NÖTH; SANTAELLA, 2014, CP 5.484, 1907).

4.2.2 Representamen, Objeto, Interpretante

Sendo uma relação triádica, o signo tem três relatos. Sendo uma relação triádica genuína, seus relatos não são intercambiáveis: cada um deles tem um papel específico.

Representamen

Uma das definições de representamen que Peirce apresenta esclarece alguns pontos a respeito do processo de semiose:

> Minha definição de representamen é como segue: um REPRESENTAMEN é um sujeito de uma relação triádica PARA um segundo, chamado seu OBJETO, POR um terceiro, chamado seu INTERPRETANTE, esta relação triádica sendo tal que o REPRESENTAMEN determina que seu interpretante esteja na mesma relação triádica para o mesmo objeto para algum [outro] interpretante. Segue-se imediatamente que essa relação não pode consistir de nenhum evento real que possa ter ocorrido; pois nesse caso deveria haver outro evento real conectando o interpretante para um [outro] interpretante seu próprio no qual o mesmo seria verdadeiro; e portanto haveria uma série infinita de eventos os quais poderiam ter realmente ocorrido, o que é absurdo. Pela mesma razão o interpretante não pode ser um objeto individual definido. A relação deve portanto consistir em um poder do representamen de determinar algum interpretante para ser um representamen do mesmo objeto (CP 1.541-542, 1903).

Isso ilustra alguns fatos: o representamen é o sujeito que tem o poder de determinar algum interpretante; não é preciso que isso ocorre realmente, basta que ocorra potencialmente. Mas, ocorrendo, pode iniciar uma série infinita de eventos semelhantes. Cria uma relação, mas essa relação não consiste de um evento "real definido". O representamen é, portanto, aquela parte do signo que tem o poder de provocar a semiose. É nesse sentido que ele é o Primeiro da relação triádica. A questão da primeiridade do primeiro relato do signo é esclarecida por Santaella (2000), citando Ransdell (1966):

> A solução está na distinção entre o primeiro termo da relação e aquilo que desempenha o papel de primeiro termo dessa relação. Qualquer coisa que seja, pode ser um signo, isto é, pode funcionar nesse papel; mas para que faça isso, deve ser algum caráter em virtude do qual assim possa funcionar. Esse caráter é o que constitui o fundamento ou razão de sua capacidade para ser um signo, embora ele não seja realmente um signo enquanto ele não for interpretado como tal (RANSDELL, 1966 apud SANTAELLA, 2000, p. 21).

Portanto, o representamen é primeiridade porque é uma potencialidade, uma possibilidade de interpretação, mesmo enquanto ainda não é efetivamente interpretado e se torna primeiro correlato de um signo.

Finalmente, Nöth e Santaella (2014) nos chama a atenção para a confusão terminológica entre esse primeiro correlato do signo e o signo como um todo; um certo descuido dos autores, inclusive de Peirce, referir-se ao representamen —ou outro nome que tenha em outra teoria— como "signo".

Objeto

Corresponderia, de forma aproximada como nos lembra Nöth e Santaella (2014), ao denotado no signo. Pode ser um objeto "perceptível, ou apenas imaginável, ou mesmo inimaginável em um [certo] sentido", segundo Peirce (CP 2.230, 1910). Incluem-se aí desde coisas singulares existentes a classes de coisas ((NÖTH; SANTAELLA, 2014)). Ainda segundo esse autor, Peirce reconheceu duas espécies de objeto: o objeto imediato e o objeto "mediato, real ou dinâmico". Nas palavras de Peirce:

> Temos de distinguir o Objeto Imediato, o qual é o Objeto como o próprio Signo o representa, e cujo Ser é portanto dependente da sua Representação no Signo, do Objeto Dinâmico, o qual é a Realidade que de algum modo conspira para compelir o Signo à sua Representação (CP 4.536, 1905).

Para Nöth e Santaella (2014) o objeto dinâmico, fora do signo, é o segmento da realidade mediato e dinâmico porque só pode ser indicado no processo de semiose. Por outro lado, o objeto imediato é como o signo o representa. A relação do signo com seus objetos é melhor ilustrada por Peirce na seguinte passagem:

> Eu defino um Signo como qualquer coisa que por um lado é determinado por um Objeto e por outro determina uma ideia na mente de uma pessoa, tal que esta última determinação, que eu denomino Interpretante do signo, é desse modo mediatamente determinada por aquele Objeto. Um signo, portanto, tem uma relação triádica com seu Objeto e com seu Interpretante. Mas é necessário distinguir o Objeto Imediato, ou o Objeto como o Signo o representa, do Objeto Dinâmico, ou Objeto realmente eficiente mas não imediatamente presente (CP 8.343, 1908).

Fica aqui claramente ilustrado o papel do Objeto Dinâmico: é ele que determina o Signo, que por sua vez o representa como Objeto Imediato. Este último tem seu ser no Signo, nunca fora dele. Fica claro também como pode ser a visão de que "tudo é signo": qualquer coisa que se apresenta à nossa mente é signo, não Objeto Dinâmico —ainda que em certas circunstâncias nos sintamos tentados a dizer que os objetos que nos cercam são, eles mesmos, que se apresentam à nossa mente. De acordo com o modo peirceano de enxergar o fenômeno, isso é falso: somente o signo tem efeito em nossa mente, não os Objetos Dinâmicos. E seu efeito é o Interpretante, como veremos a seguir.

Interpretante

Vamos nos valer do excelente resumo da noção de interpretante que nos é dada por Nöth e Santaella (2014):

> Peirce deu uma definição pragmática da significação quando definiu o interpretante como o 'próprio resultado significante', ou seja, o 'efeito do signo' (CP 5.474-475 [1898]), podendo também ser 'algo criado na mente do intérprete' (CP 8.179 [1909]). Em conformidade com sua teoria de que as ideias são signos e com a sua visão da interpretação como processo de semiose, também definiu o interpretante como signo: 'um signo dirige-se a alguém, isto é, cria na mente dessa pessoa um signo equivalente, ou talvez um signo mais desenvolvido. Chamo o signo assim criado o interpretante do primeiro signo.' (CP 2.228 [1897]).

Há mais de um: são três interpretantes. O Interpretante Imediato "consiste na *Qualidade* da Impressão que um signo está apto a produzir, não em nenhuma reação real" (8.315, 1909) ou seja, é tudo o que o signo potencialmente pode produzir sobre uma mente. O Interpretante Dinâmico é "qualquer interpretação que qualquer mente efetivamente faça de um signo" (8.315, 1909) e o Interpretante Final "não consiste no modo como nenhuma mente age, mas no modo como qualquer mente agiria" (8.315, 1909), uma tese melhor explicada em outro trecho: "é aquele que decidir-se-ia finalmente ser a interpretação verdadeira se a consideração a respeito do assunto fosse levada tão longe que atingisse uma opinião última" (8.184, 1909).

4.2.3 Classificação dos signos

Em uma carta a Lady Victoria Welby Peirce afirma que há 3^{10} ou 59.049 configurações possíveis de signo, se levarmos em conta o representamen, os dois objetos e os três interpretantes possíveis. Ele mesmo não as estudou, deixando-as "para futuros exploradores" (CP 8.343, 1908). Em uma carta anterior afirma que as principais classes de signo são dez (CP 8.341, 1904), que exporemos aqui. Divide inicialmente os signos em três tricotomias, levando em conta suas categorias universais:

> Signos são divisíveis em três tricotomias; primeiro de acordo com o signo em si mesmo [representamen] é uma mera qualidade, é um existente real [*actual*, em inglês], ou uma lei geral; segundo, de acordo com a relação do signo com seu objeto consistir no signo ter alguma característica em si mesmo, ou uma relação existencial com o objeto, ou em sua relação com um interpretante; terceiro, de acordo com seu interpretante representá-lo como um signo de possibilidade ou um signo de fato ou um signo de razão (CP 2.243, 1903).

A primeira tricotomia, que diz respeito ao representamen, revela as seguintes divisões do signo:

Qualissigno que é uma qualidade que é signo. Peirce acrescenta: "não pode agir realmente como signo enquanto não é corporificada; mas a corporificação não tem nada a ver com seu caráter como signo" (CP 2.244, 1903).

Sinsigno que é uma "coisa existente real ou evento que é signo. Só pode sê-lo através de suas qualidades; portanto envolve um qualissigno, ou melhor, vários qualissignos. Mas esses qualissignos são de um tipo peculiar e somente formam um signo por serem realmente corporificados" (CP 2.245, 1903).

Legissigno "é uma lei que é um signo. Esta lei é normalmente estabelecida pelos homens. Todo signo convencional [uma palavra, por exemplo] é um legissigno (mas não o contrário). Não é um objeto singular, mas um tipo geral que tendo sido acordado, será significante. Todo legissigno significa através de uma instância de sua aplicação, que pode ser denominado sua Réplica. [...] A Réplica é um Sinsigno. Assim, todo Legissigno requer Sinsignos. Mas estes não são Sinsignos comuns" (CP 2.246, 1903).

A segunda tricotomia estabelece-se na relação do signo com seu objeto; em uma carta de 1904 a Lady Welby, Peirce afirma que trata-se da relação do signo com seu objeto dinâmico (CP 8.335, 1904). São:

Icone é "um signo que se refere ao Objeto que denota meramente em virtude de características próprias, e que ele possui, exatamente as mesmas, quer tal Objeto exista realmente ou não" (CP 2.247, 1903). Essa definição exige um pouco de aprofundamento: trata-se de uma relação cuja característica é a primeiridade. Nöth e Santaella (2014) nos lembra que um quali-signo icônico (ícone puro) é "só uma possibilidade hipotética da existência de um signo, pois o signo genuíno participa necessariamente das categorias da secundidade (qua objeto) e da terceiridade (qua interpretante)". Portanto, prossegue o autor, um ícone puro seria um signo não comunicável. Poderia ocorrer em circunstâncias especiais que não fazem parte da semiose cotidiana. Nöth e Santaella nos auxilia a identificar os ícones quando aparecem em sua forma cotidiana, como sinsignos e legissignos icônicos:

> O critério para defini-los é o da similaridade entre representamen e objeto. Peirce fala de um signo que é 'semelhante' ao seu objeto (CP 3.362 [1885]), mas também se refere a um signo que participa 'do caráter do objeto' (CP 4.531 [1905]) e, ainda, de um signo 'cujas qualidades são semelhantes às do objeto e excitam sensações análogas na mente para a qual é uma semelhança' (CP 2.299 [c. 1895]). Os seus exemplos são de retratos, pinturas (CP 2.92 [1902]), fotografias (CP 2.280 [1895]), metáforas, diagramas, gráficos lógicos (CP 4.418-420 [1903]) e até fórmulas algébricas. Muitos desses signos não são semelhantes aos seus objetos, no sentido ordinário da palavra. Por que, por exemplo, as fórmulas algébricas e os diagramas seriam ícones? A chave da iconicidade desses signos reside na noção das correspondências relacionais. Peirce explica: 'muitos diagramas não se assemelham de modo algum aos seus objetos quanto à aparência; a semelhança entre eles consiste apenas da relação entre suas partes' (CP 2.282 [1893]) (NÖTH; SANTAELLA, 2014).

Índice é "um signo que se refere ao Objeto que denota em virtude de ser realmente afetado por esse Objeto" (CP 2.248, 1903), uma relação de secundidade. Nöth e Santaella (2014) especifica: "tais relações têm, principalmente, o caráter de causalidade, espacialidade e temporalidade".

Símbolo é "um signo que se refere ao Objeto que denota em virtude de uma lei, usualmente uma associação de ideias gerais, que operam fazendo com que o Símbolo seja interpretado como se referindo àquele Objeto. É portanto um tipo geral ou lei, vale dizer, um Legissigno. Como tal age através de uma Réplica" (CP 2.249, 1903). Nöth e Santaella (2014) especifica: é "o signo da segunda tricotomia que participa da categoria da terceiridade", situando "o hábito, a regra, a lei" na relação entre representamen e objeto.

A terceira tricotomia considera o signo "do ponto de vista da relação entre representamen e interpretante", podendo ser "rema, dicente —também chamado dicissigno— ou argumento. Essa divisão triádica 'corresponde à antiga divisão (da lógica) entre termo, proposição e argumento, modificada para ser aplicável aos signos em geral' (CP 8.337 [1904])" (NÖTH; SANTAELLA, 2014):

Rema "na lógica é ' simplesmente um nome de classe o um nome próprio'. No sentido mais geral da semiótica, um rema é, portanto, 'qualquer signo que não é verdadeiro nem falso, como quase cada uma palavra por si, exceto sim e não (CP 8.337 [1904])" (NÖTH; SANTAELLA, 2014).

Dicente é "a unidade mínima para exprimir ideias que podem ser ou verdadeiras ou falsas" (NÖTH; SANTAELLA, 2014). "A prova característica mais à mão que mostra se um signo é um dicissigno ou não é que o dicissigno é ou verdadeiro ou falso, mas não fornece as razões de ser desta ou daquela maneira" Peirce (1931-1958 apud NÖTH; SANTAELLA, 2014, CP 2.310, 1903).

Argumento "Logo que o signo supera o quadro proposicional e passa a participar de um discurso racional mais estendido, chega à categoria da terceira tricotomia. Um argumento é, portanto, 'o signo de uma lei' (CP 2.252 [1903]), 'a saber, a lei segundo a qual a passagem das premissas para as conclusões tende a ser a verdadeira' (CP 2.263 [1903])" (NÖTH; SANTAELLA, 2014).

As dez principais classes de signos

O leitor atento observará que havendo três tricotomias —signo enquanto representamen, em sua relação com o objeto e com o interpretante— e tendo cada tricotomia três classificações possíveis, deveria haver 27 (3^3) classes de signo. Mas só há dez. A

razão disso é que há situações impossíveis do ponto de vista semiótico. Por exemplo, um qualissigno, um signo que é qualidade: a relação dele com seu objeto não se dá diretamente (ou por relação causal, espacial ou temporal), nem se dá em virtude de uma lei. Portanto, não pode ser nem um índice, nem um símbolo: só pode ser um ícone. Se lembrarmos que a terceiridade pode envolver secundidade e esta pode envolver primeiridade, mas não o contrário, podemos dizer, como regra geral, que o primeiro (representamen), segundo (objeto) e terceiro (interpretante) relatos de um signo são tais que as categorias dos relatos subsequentes são iguais ou inferiores às categorias dos relatos precedentes. Assim, um

Figura 14 – Classes de signos possíveis, de acordo com Jungk (2013)

CATEGORIA	**TRICOTOMIA**		
	1ª (representamen)	**2ª (objeto)**	**3ª (interpretante)**
Primeiridade	qualissigno	ícone	rema
Secundidade	sinsigno	índice	dicente
Terceiridade	legissigno	símbolo	argumento

sinsigno só pode ser representamen de um índice ou de um ícone, mas não de um símbolo (a não ser no caso degenerado em que é uma réplica); um índice participa de um signo cujo interpretante pode ser um rema ou um dicente, mas não um argumento, e assim por diante. Todas as combinações possíveis são assinaladas pelas flechas na Figura 14. Assim, são possíveis apenas dez classes de signo, descritas na Tabela 2.

4.2.4 Tipos de raciocínio

Os tipos de raciocínio são objeto da Crítica, ou Lógica Crítica, que segundo Peirce "classifica argumentos e determina a validade e grau de força de cada tipo" (CP 1.191, 1903). Como os demais temas de sua obra, pode-se perceber o desenvolvimento de suas ideias ao longo do tempo. De acordo com Peirce, há três tipos de argumento: Dedução, Indução e Abdução (CP 2.96, 1903).

A Dedução, também chamada por Peirce de Argumento Obsistente, é um "argumento representando fatos nas Premissas, tais que quando os representamos em um Diagrama nós nos achamos compelidos a representar o fato declarado na Conclusão" (CP 2.96, 1903). Ou seja, é um argumento —o único, segundo Peirce— compulsório, querendo com isso dizer que as Premissas são um Índice do fato compelido, ou seja, a Conclusão. É por essa relação direta que Peirce o chama de Obsistente. Segundo ele, todas as de-

Tabela 2 – As dez principais classes de signo

Representamen	Objeto	Interpretante	Características	Exemplo
Qualissigno	Icônico	Remático	qualidade que é um signo	sensação de uma cor ou de um sabor
Sinsigno	**Icônico**	Remático	signo concreto, que representa seu objeto por suas qualidades	exemplar individual de um mapa ou diagrama
Sinsigno	**Indicial**	**Remático**	dirige a atenção a um objeto concreto pela sua mera presença	grito espontâneo, por exemplo, de surpresa ou de dor
Sinsigno	Indicial	**Dicente**	é afetado diretamente por seu objeto e dá informações sobre ele	cata-vento, termômetro
Legissigno	**Icônico**	Remático	lei representada iconicamente	placa de trânsito "pedestres" quando mostrada na apostila; diagrama da hélice do DNA; qualquer onomatopeia
Legissigno	**Indicial**	**Remático**	cada um de seus casos é realmente afetado por seu objeto, e atrai atenção para esse objeto	pronome demonstrativo
Legissigno	**Indicial**	**Dicente**	lei geral afetada por um objeto real, de tal modo que forneça informação definida a respeito desse objeto	pregão de vendedor de rua; placa de trânsito específica na rua; comando militar
Legissigno	**Simbólico**	**Remático**	signo convencional que não tem o caráter se uma proposição	qualquer substantivo
Legissigno	Simbólico	**Dicente**	combina símbolos remáticos em uma proposição	qualquer proposição completa
Legissigno	Simbólico	**Argumento**	signo do discurso racional	silogismo

Fonte: adaptado de Nöth e Santaella (2014)

monstrações de Euclides são desse tipo: a relação entre Premissas e Conclusão é a da Secundidade (CP 2.96, 1903).

A Abdução, ou Argumento Originário, segundo Peirce, é um argumento que "apresenta fatos em suas premissas que apresentam uma similaridade com o fato enunciado na Conclusão" (CP 2.96, 1903). Essas similaridade, entretanto, não nos obriga a admitir a Conclusão como verdadeira, mas somos "apenas inclinados a admiti-la como representando um fato do qual os fatos da Premissa constituem um Ícone" (CP 2.96, 1903). Chama-se Originário pois "é o único tipo de argumento que inicia uma nova ideia" (CP 2.96, 1903); a relação entre as Premissas e a Conclusão é da natureza da Primeiridade.

A Indução é o Argumento Transuasivo. É um argumento que "emerge de uma hipótese, resultante de uma Abdução anterior, e de predições virtuais, sacadas por Dedução, dos resultado de possíveis experimentos, e tendo realizado os experimentos, conclui que a hipótese é verdadeira na medida em que aquelas predições se verificam, mantendo-se esta conclusão, no entanto, sujeita a prováveis modificações que se seguiriam a futuros experimentos" (CP 2.96, 1903). É o argumento mais complexo: pressupõe uma hipótese cuja experimentação verifica, mas não de maneira obrigatória (caso em que seria uma Dedução). As Premissas são um Símbolo da Conclusão, na medida em que esta última depende da preditibilidade das primeiras, que têm por isso o caráter de lei. Por isso é Transuasivo: a relação entre Premissas e Conclusão é da natureza da Terceiridade.

Uma exemplo nos foi dado em sala de aula pelo professor Jorge de Albuquerque Vieira: "dedução: temos em mãos um saco opaco cheio de bolas que sabemos serem todas da mesma cor; tiro uma bola do saco, vejo que é vermelha, e por dedução concluo que todas são vermelhas. Indução: temos em mãos um saco opaco cheio de bolas; tiro uma bola do saco, vejo que é vermelha; tiro a segunda, a terceira, até a décima, são todas vermelhas; por indução concluo que todas devem ser vermelhas. Abdução: tenho em mãos um saco cheio de bolas vermelhas; vejo no chão uma bola vermelha; abdutivamente concluo que a bola pertence ao saco que tenho em mãos. É uma hipótese a ser verificada".

4.2.5 Metodêutica

A metodêutica, apesar de sua importância, é o ramo menos estudado da semiótica peirceana, e uma das razões é que grande parte dos escritos de Peirce a esse respeito permanece sem publicação (SANTAELLA, 1999, p. 379).

A terminologia peirceana referia-se a esse ramo inicialmente como "Retórica Especulativa", tendo o termo "Metodêutica" substituído o anterior a partir de aproximadamente 1900 (cf. SANTAELLA, 1999, p. 389). Em um de seus textos Peirce a define como segue:

> Lógica Transuasional, que eu denomino Retórica Especulativa, é substancialmente o que é chamado de metodologia, ou melhor, de *metodêutica*. É a doutrina das condições gerais da referência de Símbolos e outros Signos aos Interpretantes que eles almejam determinar... (CP 2.93, 1902).

Seu escopo é, portanto, "uma teoria dos interpretantes responsável pela futuridade dos signos, o que significa uma teoria do poder gerativo do signo de se transformar em outro signo. Assim sendo, a retórica especulativa lida com a vida dos signos após serem percebidos por uma mente. Lida com os diferentes tipos de desenvolvimento que vários tipos de signo experimentam no processo de recepção" (SANTAELLA, 1999, p. 380–381).

Pode parecer que há uma sobreposição entre esse escopo e o da lógica crítica, que lida com os tipos de argumento, mas não. A retórica especulativa está preocupada com o uso eficiente dos signos, no qual "há um elemento de prescrição, e este pertence ao escopo da retórica" (SANTAELLA, 1999, p. 381). Seu objeto de estudo são os Interpretantes, com ênfase nos Interpretantes Finais (SANTAELLA, 1999, p. 384).

O estudo da metodêutica se baseia em uma classificação dos signos baseada não em três tricotomias, como vimos, mas em dez tricotomias. A exata classificação dos signos de acordo com essas dez tricotomias ainda é assunto de debate, e não entraremos em detalhes. Contentamo-nos em citá-las com base em Santaella (1999), p. 381–382:

1. De acordo com a natureza do signo: qualissigno, sinsigno e legissigno. Essa primeira tricotomia também pode aparecer com respeito ao modo da possível apresentação do signo: potissigno, atissigno, famissigno.
2. De acordo com o modo de apresentação do objeto imediato: descritivo, nominativo, copulante.
3. De acordo com a natureza do objeto dinâmico: abstrativo, concretivo, coletivo.
4. De acordo com a relação do signo com o objeto dinâmico: ícone, índice, símbolo.
5. De acordo com a natureza ou modo de apresentação do interpretante imediato: hipotético, categórico, relativo.
6. De acordo com a natureza do interpretante dinâmico: simpatético, chocante, usual.
7. De acordo com a relação do signo com o interpretante dinâmico ou sua maneira de apelar para o intepretante dinâmico: sugestivo, imperativo, indicativo. Essa tricotomia pode também aparecer como: ejaculativa, imperativa, significativa.
8. De acordo com a natureza ou propósito de interpretante final ou eventual: gratificar, produzir ação, produzir auto-controle.
9. De acordo com a relação do signo com o interpretante final: rema, dicente, argumento. Essa tricotomia pode também aparecer de acordo com a influência do signo: sema, fema, deloma.
10. De acordo com a relação triádica do signo ou como a natureza da certeza da locução: certeza de instinto, certeza de experiência, certeza de forma.

Exporemos a título de exemplo a análise da oitava tricotomia dada por Santaella (1999): há, segundo Peirce, "três tipos de propósito que guiam a tendência do signo em direção a um estado final". Santaella enumera:

- o propósito estético do interpretante final, que leva o signo a produzir qualidades de sentimento que são admiráveis e resultam da abdução.

- o propósito de produzir ação, nos chamados interpretantes finais práticos. Há vários graus de interpretantes práticos, mas qualquer que seja seu tipo a ação sempre terá uma certa orientação ética, que é uma característica dos interpretantes que são voltados à direção da conduta.

- o propósito de uma mudança de hábito, quando a direção do interpretante final será pragmática. Essa mudança de hábito, através do auto-controle crítico e deliberado que exercemos sobre nossas crenças são orientados principalmente pelos princípios guias da lógica.

De qualquer modo, a evolução da retórica especulativa levou Peirce à metodêutica, que, em última análise, leva à lógica de como a investigação deve ser conduzida. Peirce deu importância crescente a esses estudos, inclusive os métodos de descoberta, de resolução de problemas e da própria investigação, a partir de 1900. Aparentemente Peirce defendia a ideia de que há uma lógica da descoberta, "tirando essa questão do domínio exclusivo da psicologia" (SANTAELLA, 1999, p. 393). Essa lógica da descoberta leva à prescrição do uso adequado dos três tipos de raciocínio —dedutivo, indutivo e abdutivo— para a criação e construção das teorias científicas que, segundo Santaella, eram para Peirce "a maior façanha prática da lógica e, no caso das ciências físicas. 'o mais perfeito exemplo de aplicação exitosa de pensamento ao mundo externo' (CP 7.276 [sem data])" (SANTAELLA, 1999, p. 384).

4.3 Algumas considerações adicionais

A semiótica de Peirce presta-se à reflexão a respeito dos mais diversos fenômenos, tornando possível o embasamento de considerações que de outro modo não teriam o alcance que encontraram.

4.3.1 Os diagramas no raciocínio

Peirce defendia que todo raciocínio necessário era diagramático; por essa razão desenvolveu seus Grafos Existenciais, uma maneira gráfica de fazer inferências lógicas. De acordo com ele o raciocínio diagramático se dava assim:

> Construímos um ícone do nosso estado hipotético de coisas e passamos a observá-lo. Essa observação nos leva a suspeitar que alguma coisa é verdadeira, o que podemos ou não ser capazes de formular com precisão, e passamos a inquirir se isso [a suspeita] é ou não verdadeiro. Para

> esse propósito é necessário formar um plano de investigação e essa é a parte mais difícil de toda a operação. Não temos de apenas selecionar as características do diagrama que serão pertinentes prestar atenção, mas também é de grande importância voltar repetidas vezes a certas características. De outro modo, ainda que nossas conclusões possam ser corretas, não serão as conclusões particulares que almejávamos. Mais o maior ponto da arte consiste na introdução da abstrações adequadas. Quero com isso dizer tal transformação dos nossos diagramas que características de um diagrama possam aparecer em outros como coisas (CP 5.162, 1903).

Peirce julgava possível criar, com base em suas ideias, uma teoria para o planejamento de demonstrações matemáticas. Claro, não a aplicava somente aos raciocínios necessários (deduções). Em outro trecho utiliza outra forma de expressar o uso de diagramas:

> [O] processo de abstração é, ele próprio, um tipo de observação. A faculdade que eu chamo de observação abstrativa é uma que as pessoas comuns reconhecem perfeitamente, mas para a qual as teorias dos filósofos raramente abre espaço. É experiência familiar a todo ser humano desejar algo além dos seus presentes meios, e em sequência ao desejo perguntar-se, 'Desejaria isso do mesmo modo, se tivesse os meios de possuí-lo?'. Para responder à questão, ele vasculha seu coração, e ao fazê-lo faz o que chamo de observação abstrativa. Ele cria em sua imaginação um tipo de esqueleto de diagrama ou um esboço do contorno de si mesmo, considera quais modificações o estado hipotético de coisas exigiria que se fizesse na figura, e então a examinaria, ou seja, observaria o que imaginou, para ver se o mesmo desejo ardente está lá para ser discernido. Através desse processo, que é no fim muito parecido ao raciocínio matemático, podemos atingir conclusões sobre o que poderiam ser verdades sígnicas em todos os casos, desde que a inteligência as usando fosse científica (CP 2.227, 1897).

Observemos que é um tratamento observacional das abstrações. Uma aplicação interessante da semiótica peirceana seria ao raciocínio matemático, à luz do conhecimento que surgiu depois dele, entre eles as noções de computabilidade e funções computáveis, abordadas no presente trabalho, em 2.2.

4.3.2 Semiose de máquinas

Inúmeros autores já se debruçaram e se debruçam sobre o assunto (NÖTH, 2007; FETZER, 2001; STEINER, 2013; SKAGESTAD, 1996); debruçar-nos-emos sobre algumas considerações de Peirce. Algumas máquinas de seu tempo eram "máquinas lógicas": aparelhos mecânicos que permitiam que se configurasse um conjunto de premissas e, ao mover de uma alavanca, apresentavam as conclusões lógicas possíveis. Peirce escreveu em 1887 um artigo (PEIRCE, 1887) a respeito, publicado no *American Journal of Psychology*.

Nesse artigo apresenta duas máquinas lógicas, explicando como eram configuradas as premissas e obtidas as conclusões. Passa então a considerações a respeito:

> O segredo de todas as máquinas raciocinantes é no fim das contas muito simples. É que qualquer relação entre objetos raciocinados destinada a ser a conexão de um raciocínio, a mesma relação geral tem de ser capaz de ser introduzida entre certas partes da máquina (PEIRCE, 1887, p. 168).

O exemplo que dá é o do silogismo: se A então B, se B então C, portanto, se A então C. Diz Peirce que é preciso ter uma conexão a ser introduzida na máquina tal que se o evento A ocorrer na máquina, o evento B também ocorre; e o mesmo entre B e C. Dessa forma, produz-se virtualmente uma conexão entre o evento A e o evento C. Algo mais interessante vem a seguir:

> Esse é o mesmo princípio que repousa no fundamento de qualquer álgebra lógica; só que na álgebra, ao invés de depender diretamente das leis da natureza, estabelecemos regras convencionais para as relações usadas. Quando performamos um raciocínio usando somente nossa mente fazemos substancialmente a mesma coisa, vale dizer, construímos uma imagem em nossa fantasia sob certas condições gerais, e observamos o resultado. Nesse ponto de vista, também, toda máquina é uma máquina raciocinante, tanto quanto haja certas relações entre suas partes, cujas relações envolvem outras relações que não são expressamente intencionadas (PEIRCE, 1887, p. 168).

Isso se aplica, naturalmente, aos computadores, tanto os modernos quanto os de arquitetura estritamente von Neumann. Peirce estende seu raciocínio para máquinas cujo funcionamento não depende das leis da mente humana, mas apenas da "razão objetiva incorporada nas leis da natureza" (PEIRCE, 1887, p. 168). Nesse sentido, um aparato físico ou químico utilizado para realização de experimentos é também uma máquina raciocinante.

Peirce prossegue apontando agora as coisas que essas máquinas raciocinantes não são capazes de fazer: em primeiro lugar, são destituídas de toda originalidade, de toda iniciativa. "Não encontra seus próprios problemas, não se alimenta sozinha", diz Peirce. Mas adiciona que "isso não é um defeito da máquina; não queremos que cuidem de seus próprios negócios, mas dos nossos" (PEIRCE, 1887, p. 169). E em segundo lugar, segundo Peirce, "a capacidade da máquina tem limitações absolutas; foi criada para fazer uma certa coisa, e não pode fazer nada mais" (PEIRCE, 1887, p. 169). Dá como exemplo o fato de as máquinas estudadas poderem lidar com um número limitado de silogismos. E, embora o mesmo aconteça com a mente sozinha, "a mente trabalhando com lápis e bastante papel não tem esses limites" (PEIRCE, 1887, p. 169). Deixou, obviamente, a

ressalva de que esses limites tendem a ser rompidos, como de fato o foram nos modernos computadores.

Conclui o artigo afirmando que não vê grande dificuldade na ampliação do número de termos que essas máquinas podem tratar, e sugerindo que um bom paradigma de funcionamento seria o tear de Jacquard. Nesse tear os padrões do tecido são descritos através de cartões perfurados, à moda dos primeiros computadores.

4.3.3 Significado e vagueza em Peirce

A semiótica peirceana realiza-se num corpo filosófico extenso que ainda não foi completamente explorado. Nele é possível encontrar uma visão lúcida do processo de significação, cuja descrição completa está além do escopo do presente trabalho e do alcance deste autor. Para apresentar o assunto resumiremos pontos de Nöth e Santaella (2011), que nos dão uma visão aguda do tema, inciando por uma citação de Peirce em Brier (2008):

> Nenhum pensamento atual presente (o qual é uma mera sensação) tem qualquer significado, qualquer valor intelectual; isso tem base não no que é realmente pensado, mas com o que esse pensamento pode estar conectado em representação por pensamentos subsequentes; tal que o significado de um pensamento é de resto algo virtual. Pode-se objetar que se nenhum pensamento tem significado, todo pensamento [tudo o que se pensa] é sem significado. Mas isso é uma falácia similar a dizer que se em nenhum dos espaços sucessivos que um corpo preenche há espaço para movimento, então não há espaço para movimento através do todo. Em nenhum instante do meu estado mental há cognição ou representação, mas na relação dos meus estados mentais em diferentes instantes há (CP 5.289, 1868).

Nöth e Santaella apontam o enigma: primeiro, pensamentos e por extensão signos em geral aparentemente não têm significado sempre; segundo, o significado de um pensamento é "de resto algo virtual". Há dois sentidos em que um pensamento não tem significado. Num, é que há signos que interpretados pelo pensamento não tem significado próprio: são os signos indexicais —os índices. Eles não significam, mas transmitem informação, no sentido em que o termo é usado nas ciências naturais e da informação. O outro sentido é o endereçado na citação: o pensamento não tem significado quando é uma mera sensação. "Pensamento atual presente" não tem significado uma vez que ainda não está conectado com outros pensamentos, que ele poderia representar. Os autores notam que "Peirce distingue o pensamento do ato de pensar. O último tem significado sempre, uma vez que o ato de pensar, em contraste com o pensamento, é sempre dialógico". E citam Peirce novamente:

> Todo ato de pensar tem forma dialógica. O si-mesmo de um instante apela para assentimento do si-mesmo mais profundo. Consequentemente, todo ato de pensar é conduzido em signos que são principalmente da mesma estrutura geral das palavras; aqueles que não o são, são da natureza daqueles signos dos quais temos necessidade de vez em quando em nossa conversa um com o outro para suprir os defeitos das palavras, ou símbolos. Esses pensamentos-signo não-simbólicos são de duas classes: primeiro, figuras ou diagramas ou outras imagens (eu os chamo de Ícones) tais quais devem ser usados para explicar as significações das palavras; e segundo, signos mais ou menos análogos a sintomas (eu os chamo Índices) dos quais as observações do entorno através das quais sabemos sobre o que alguém está falando são exemplos. Os Ícones principalmente ilustram as significações dos pensamentos-predicado, os Índices as denotações dos pensamentos-sujeito. A substância dos pensamentos consistem dessas três espécies de ingrediente (CP 6.338, 1908).

Portanto, o significado verbal (que Peirce chama de significação no texto) necessita de ícones. Mas em Peirce o significado pode não se incorporar, pode ser virtual, como vimos acima. Pode ser um *esse in futuro*, como nos lembram Nöth e Santaella citando (CP 2.148, 1902). Essa dupla natureza do signo é assim explicada pelos autores: a incorporação é fenômeno de secundidade. Assim, para transmitir seu significado em uma instância específica de semiose, o signo precisa estar incorporado para criar seu efeito semiótico na forma de um interpretante. Mas logicamente o signo não é fenômeno de secundidade mas de terceiridade, e como tal não precisa estar incorporado numa instância específica de uso. Como tal, o signo não tem ser, apenas necessidade lógica.

Mas significado não é um conceito chave na semiótica peirceana e não tem um lugar sistematicamente determinado nela, segundo os autores. É principalmente uma questão para o interpretante, mas também pertence ao objeto do signo. No caso de um símbolo, por exemplo, o seu objeto imediato é o nosso conhecimento colateral do objeto dinâmico do símbolo: é o significado transmitido pelo signo, e é complementar ao interpretante. Esse sentido é referido por Peirce na seguinte definição: "Um signo representa algo para a ideia que ele produz, ou modifica. Ou, é um veículo transmitindo para dentro da mente algo de fora. Aquilo que ele representa é chamado seu objeto; o que ele transmite, seu significado; e a ideia para a qual ele dá origem, seu interpretante" (CP 1.339, sem data).

Finalmente, os autores consideram a questão da vagueza trazendo-nos o inesperado —e não obstante coerente— ponto de vista peirceano, no qual a vagueza é uma "necessidade semiótica para o crescimento dos signos na troca conversacional, na pesquisa científica, e na linguagem e cultura em geral. [...] Indeterminação e vagueza são inerentes à representação e interpretação. Nos diálogos, não é apenas impossível mas também indesejável ser absolutamente preciso" (NÖTH; SANTAELLA, 2011, resumo). Uma chave para a teoria da vagueza de Peirce é a doutrina do sinequismo, citada na página 86.

Lembrando-nos inicialmente que "o sonho de uma linguagem livre de ambiguidades e vagueza tem sido acalentado desde que Francis Bacon fez seu apelo por uma linguagem livre de sentenças que 'impõem sobre nós [aquelas] falsas aparências' ", prosseguem apontando diversos autores que estudaram o tema, alguns também considerando a vagueza um mal a ser extirpado. A teoria da vagueza de Peirce foi reconstruída graças ao trabalho de diversos autores que também mostraram que a lógica de Peirce resolve alguns problemas fundamentais no tema do significado vago. O próprio Peirce chegou a criticar a ausência de atenção sobre o tema de lógicos seus contemporâneos.

Seguem: para Peirce, indeterminação é parte tanto da vagueza quanto da generalidade, e estas são antíteses análogas entre si: o geral é "aquilo para o que não se aplica o princípio do terceiro excluído. Um triângulo em geral não é nem isósceles nem equilátero; nem [...] escaleno" (CP 5.505, 1905). O vago, em contraste, é "aquilo sobre o que o princípio da contradição não se aplica. Pois não é falso que um animal (num sentido vago) é macho nem que é fêmea" (ibid.). Do ponto de vista retórico, elocuções vagas e gerais oferecem a oportunidade de tornar palavras e argumentos mais precisos no decorrer de uma conversa, se desejado; e abrem espaço para interpretações imaginativas: " 'O homem é mortal.' 'Qual homem?' 'Qualquer um que você escolher' " (ibid.).

Os autores informam, citando Tiercelin (2005), que a vagueza é inerente à percepção e à cognição porque interpretamos nossas percepções com base em crenças e hábitos de julgamento, que são imprecisos: quando dizemos a cor de um objeto o fazemos por hábito, não através de uma cartela de cores. Uma vez que as cores formam um contínuo, o hábito de nomeá-las resulta em vagueza. Isso resulta na incerteza de um intérprete quanto ao significado pretendido por um falante cuja linguagem indeterminada permite várias interpretações —o que não se deve ao conhecimento incompleto do falante a respeito do estado de coisas, mas é intrínseco ao hábito da linguagem indeterminada.

Prosseguem apontando a vagueza no signo: Peirce considera suas próprias definições insatisfatórias, denotando a dificuldade essencial de se chegar a uma definição precisa; a vagueza no objeto, que depende da classe do signo: o símbolo é vago por sua generalidade, o índice, sendo vazio de significado é intrinsecamente vago, e o ícone é o signo mais vago, já que a vagueza é essencial aos fenômenos de primeiridade pois envolvem qualidades, acaso, o indeterminado, o incerto, o sentimento, o espontâneo, frescor, originalidade, hipótese e conjeturas; e finalmente a vagueza no interpretante, decorrente de sua natureza dialógica.

Finalmente apresentam as consequências da vagueza essencial dos signos, que não podem ser nem tão vagos a ponto de serem sem sentido, nem tão precisos que não possam crescer. A esse respeito concluem: "uma linguagem que é absolutamente precisa não pode mudar sem se tornar menos precisa, e uma linguagem que não muda mais só pode ser uma

linguagem morta". Não podemos terminar este resumo sem lembrarmos a imutabilidade das ditas "linguagens" de programação.

5 Semiótica de Peirce em "Semiotics of Programming"

Tendo exposto o livro de Tanaka-Ishii no cap. 3, e conceitos da semiótica peirceana no cap. 4, passamos agora à análise da aplicação que a autora dá a ela. Consideramos a seguir a abordagem que a autora dá ao tema, as consequências dessa abordagem no trabalho e, por último, fazemos considerações sobre alguns caminhos que o livro poderia ter percorrido.

5.1 Abordagem adotada

Chama a atenção que a autora julgue que a teoria semiótica não esteja suficientemente estabelecida para ser adotada de maneira direta numa introdução ao tema no seu livro (ver p. 45). Se levarmos em conta a indeterminação do quê exatamente é semiótica apontada por Nadin (2011) (ver 4.2, primeiro parágrafo) e observarmos o corpo teórico que Tanaka-Ishii analisou, no qual mesmo cientistas renomados na área identificam uma certa "confusão babilônica" (ver p. 51), não poderemos deixar de lhe dar alguma razão. Entretanto, diante da indeterminação encontrada, Tanaka-Ishii adota a postura de tentar encontrar, ela mesma, um denominador comum entre as teorias semióticas que considera mais relevantes, e o faz de uma maneira que podemos considerar eclética. E ao fazê-lo omite apresentar —e consequentemente compreender— uma base semiótica teórica; aparentemente adota conceitos de uma e outra teoria à medida em que lhe parecem necessários (ver p. 45) de um modo "cartesiano", ou seja, toma o que lhe parece indubitável e prossegue deduzindo conclusões a partir daí. A seguir, como esse processo se desenrola.

5.1.1 O que é signo?

A autora, ao omitir a apresentação de uma base semiótica teórica, omite necessariamente a definição do que vem a ser um signo para ela, embora defina já na introdução o que é uma linguagem: uma "relação de elementos linguísticos e seus significados" (ver p. 44). Logo no *Capítulo 2 - Signos computacionais em programas* apresenta os "quatro tipos de signo" (ver p. 49) que aparecem nas linguagens de computador sem sequer introduzir o conceito de signo. Nesse momento podemos apenas especular que ela entende que os signos são, ao menos nos programas de computador, as diversas "palavras"[1] que podem

[1] O termo vai entre aspas pois não faz parte da definição formal de uma linguagem de programação (ver p. 38).

compor um programa. Até o momento parece que para a autora o próprio programa não é um signo, o que, se confirmado, já é suficiente para deduzir que sua abordagem semiótica não é peirceana. De fato, no *Capítulo 3 - A confusão babilônica*, quando se espera que ao definir modelos de signo a autora defina também o que é signo, isso não ocorre propriamente: são apresentadas as definições que estão nas origens de cada modelo e começa a especulação sobre qual modelo explica melhor o funcionamento do signo. É assim, através do entendimento que ela tem do funcionamento do signo, que entendemos que para Tanaka-Ishii o signo é algo que traz alguma ideia à mente do intérprete —uma visão que não corresponde exatamente à peirceana.

O fato de a análise feita pela autora ao longo do livro se restringir, como declarou, aos signos que são variáveis nos programas (ver p. 49) diminui o impacto da ausência de um conceito amplo de signo. Entretanto, os sinais que vimos nos levam a crer que sua visão de signo se aproxima mais da visão saussureana: nem programas, nem sistemas de signos são tratados como signo, e a ação do signo é estabelecer uma relação com um significado. Supondo verdadeiro que para a autora o conceito de signo se limita ao conceito saussureano linguístico de signo, cabe-nos avaliar como a visão peirceana de signo se encaixa em seu trabalho.

Nesse ponto cabe lembrar (ver 4.2) que a teoria de Peirce é uma teoria sígnica do conhecimento apoiada nas categorias peirceanas de primeiridade, secundidade e terceiridade; só nessa moldura faz sentido pensar sua semiótica. O fato de ser um modelo triádico é quase acidental, uma consequência de suas categorias; pode-se imaginar inúmeros outros modelos triádicos que não correspondem à teoria peirceana. Por exemplo, a visão saussureana da autora, na qual há um elemento chamado significante, um outro chamado significado, e onde o efeito do signo se dilui no sistema de signos (ver 3.2.1), portanto três elementos, encaixa-se numa visão triádica. De modo quase casual Tanaka-Ishii associa cada um desses elementos a um elemento do modelo peirceano, tomando o cuidado de também apontar em que lugar ficaria o "mundo real" em ambos os casos.

Por conta disso cabe dizer que o trabalho, ao propor a composição dos dois modelos, de fato somente utiliza uma nomenclatura peirceana para renomear os elementos da semiose como são entendidos na escola de Saussure. Ou seja: a nomenclatura de Peirce é utilizada numa semiótica que não é peirceana. Podemos também inferir que a autora compartilha uma visão ontológica dualista, que opõe mente e matéria.

5.1.2 Pensando cartesianamente chega-se a uma teoria eclética

Outra característica da abordagem da autora aparece ao longo de todo o livro: partindo de um pequeno conjunto de premissas que julga corretas, segue desenvolvendo

deduções cada vez mais complexas sem se preocupar em cotejá-las com a literatura existente. É assim que chega à conclusão, no *Capítulo 3 - A confusão babilônica*, que sua análise da "correspondência" entre os modelos diádico e triádico pode contribuir com o estudo da semiótica. Essa característica lembra-nos bastante uma característica que apontamos (ver p. 84) nos corpos de conhecimento baseados no método cartesiano, a supervalorização do encadeamento lógico nos raciocínios, em detrimento da plausibilidade da conclusão.

Por esse expediente surgem, ao longo do livro, considerações sobre questões difíceis de conciliar com a semiótica. São questões que evidentemente fazem ou fizeram parte do trabalho da autora em ciência da computação e para as quais tentou criar, de uma maneira cartesiana, uma explicação utilizando termos que pertencem à semiótica numa aplicação que por vezes os desloca de seu sentido original. É assim que as questões ontológicas de "ser" e "fazer" —que escapam totalmente da semiótica— de repente são aglutinadas ao modelo semiótico diádico e triádico no *Capítulo 5 - Ser e fazer em programas*, e no *Capítulo 8 - Uma instância versus A instância* questões de generalidade e individualidade, caras a Peirce, são ignoradas no julgamento da instanciação provocada por software.

Há outros exemplos da utilização entusiasmada do método cartesiano: no *Capítulo 4 - Casamento do significante e do significado*, por exemplo, chama a atenção que a autora julgue ser um paradoxo que significante e significado sejam indissociáveis e ao mesmo tempo sua ligação seja arbitrária. De um modo inteligente, aplica os conceitos de significante e significado ao cálculo-λ e extrai daí um mecanismo de semiose que, embora seja utilizado para explicar diversas questões que surgem no livro —como algumas das que aparecem no *Capítulo 10 - Signo e tempo*—, a rigor trata-se de utilizar uma nomenclatura saussureana a conceitos deduzidos cartesianamente pela autora. Mais um desses exemplos aparece no *Capítulo 7 - Três tipos de conteúdo em programas*, no qual a autora encontra uma forma de reduzir programas a três tipos de elementos e, por serem três, "justificam" a existência da terceiridade peirceana.

Outra consequência da "liberdade" cartesiana com os conceitos que parecem corretos é que as conclusões da autora são difíceis de aproveitar em estudos semióticos, quer por estudiosos da linha saussureana, quer pelos da linha peirceana. O híbrido que se formou, embora tenha sido criado em torno das características da programação que a autora considera relevantes, não ajuda a avançar o estudo semiótico propriamente dito da programação de computadores.

Um efeito colateral desta abordagem para o nosso trabalho é que não se tem uma visão efetivamente peirceana da programação de computadores. Isso gerou, de acordo com o nosso entendimento peirceano, lacunas de duas naturezas: lacunas na compreensão

do escopo e lacunas na análise dos fenômenos observadas. As primeiras nos privaram da análise de fenômenos importantes dentro do escopo proposto, e as segundas resultam numa análise apenas parcial dos fenômenos identificados como importantes. Além disso, conceitos peirceanos são utilizados de forma incorreta: funcionam mais como um "rótulo alternativo" para fenômenos analisados sob uma óptica dualista cartesiana.

5.2 Lacunas na compreensão do escopo

A seguir apresentamos alguns elementos do fenômeno da programação de computadores cuja análise poderia ter enriquecido ainda mais a obra de Tanaka-Ishii, e que acreditamos que teriam sido levados em conta numa abordagem baseada na semiótica peirceana. Evidentemente não pretendemos que a lista seja completa.

5.2.1 Contexto da programação de computadores

O esforço de criação de software não é um esforço de programação somente (ver 2.5). Determinar o que vai ser programado não é uma tarefa trivial, exceto em casos muito simples. Os programas de computador mais complexos são gerados a partir de especificações que podem ter características semióticas dignas de nota. Além disso, o próprio processo de criação de um programa possui uma dinâmica cuja análise semiótica certamente enriqueceria a obra, mesmo sendo essa voltada para níveis semânticos (ver p. 49) que não levam em conta o contexto.

Uma outra lacuna que não se pode deixar de observar nesse nível é o contexto em que a própria autora considera sua obra. Inferimos, a partir do conteúdo do *Capítulo 8 - Uma instância versus A instância*, que o contexto em que os programas se inserem são dentro de uma moldura de utilização do computador para narrativas em linguagens naturais. De modo geral, ousamos afirmar que, uma vez que a semiótica de Peirce é uma teoria da cognição, não só o livro se enriqueceria, como o próprio trabalho da autora talvez tivesse a ganhar com a apreciação mais profunda da obra de Peirce.

5.2.2 O programa como signo e o problema da decisão

Em nossa opinião uma abordagem peirceana da programação de computadores não poderia prescindir da análise da semiose do programa em si como signo, mesmo quando se exclui o contexto em que está inserido. Como vimos em 2.4 e em 2.5.2, o programa de computador tem o duplo papel de prescrever uma função pretendida e permitir que a partir dele se crie um software. O programa é portanto um signo que sofre diversas semioses, em dois momentos principais:

- No momento da programação, temos um caso de raciocínio que se assemelha ao raciocínio diagramático que vimos em 4.3.1. Aqui o desafio do programador é identificar qual conjunto de sequencias válidas (dentro da linguagem de programação) descrevem o diagrama que representa a função pretendida pelo programa. É um tipo de raciocínio que possivelmente guarda semelhanças com o raciocínio matemático utilizado na criação de demonstrações.
- No momento da compreensão do programa, quer por seres humanos, quer por máquinas. Há aqui duas semioses: a humana, que quer entender o que faz o programa escrito, e a da máquina, que cria o software a partir do programa. Segundo Peirce (ver 4.3.2), são semioses muito semelhantes entre si.

Admitindo corretas as considerações sobre ambos os momentos, temos uma consequência semiótica que mereceria atenção, a saber:

1. Qualquer programa não-trivial é criado a partir de um raciocínio diagramático. Em algum momento de sua especificação alguém cria um ícone mental da função pretendida e a partir desse ícone prescreve ou o programa em si ou um texto a partir do qual o programador pode estritamente deduzir o programa. O processo de observação desse diagrama é da natureza da primeiridade.
2. O programa em si pode ser transformado em software e este submetido à execução. Essa transformação é da natureza da secundidade. De fato, quase todas as ações do software podem ser deduzidas a partir do texto do programa —os programas em que isso não acontece escapam do conceito de algoritmo e de computação e, embora possíveis, são indesejados; os programadores se esforçam para que todas as ações do software sejam dedutíveis a partir do programa, e é considerado inadequado o programa em que isso não acontece.
3. Portanto, o comportamento do software gerado por um programa adequadamente escrito é dedutível, sendo essa dedução da natureza da secundidade.
4. A consequência é que a partir do programa adequadamente escrito é impossível deduzir o diagrama inicial utilizado na sua geração, não importando a linguagem de programação em que seja escrito. A simples leitura, por um ser humano, do texto do programa pode induzir à formação de algum diagrama mental, mas nada garante que esse diagrama gerado a partir dessa leitura tem alguma correspondência com o diagrama inicial que gerou a especificação do programa.

Portanto, do ponto de vista semiótico é impossível, dado um programa, inferir se o software derivado dele cumpre a função pretendida que em algum momento foi pensada como um

diagrama. Para que essa inferência ocorresse seria preciso uma forma de ter acesso ao diagrama inicial que foi utilizado na prescrição do programa. Esse acesso, entretanto, não pode ser feito a partir do programa através de inferências da natureza exclusiva da secundidade, ou seja, não pode ser feito por um outro programa. Isso frustra, de certa forma, qualquer intenção de procurar uma "especificação semântica consistente" para os programas (ver p. 37, na qual citamos Sebesta (2012)) —intenção de resto já frustrada, ao menos na óptica peirceana e dentro do escopo de abrangência das linguagens naturais, pela vagueza intrínseca destas últimas (ver 4.3.3). Chama a atenção, também, a semelhança desse resultado com o de Turing (1936), que afirma que não há procedimento computável para julgar a efetividade de um procedimento computável.

Observemos também que esse resultado não contradiz Hoare (1969), apresentado na página 40: determinar os valores que as variáveis do programa devem assumir para cumprir a função pretendida corresponde a criar um diagrama mental da função pretendida e descrevê-lo sob a forma de valores de variáveis em programas —esse passo corresponde à inferência baseada na primeiridade. A descrição da função pretendida sob a forma de valores de variáveis pode ser utilizada na dedução —secundidade— de um programa.

No momento podemos somente especular, mas eventualmente casos de sucesso ou fracasso na construção de software (ver 2.5.1) podem estar associados aos diagramas icônicos utilizados na especificação deste.

5.2.3 Os signos num programa

A autora indica as variáveis, ou identificadores, como os principais signos dignos de nota nos programas de computador. São, de fato, signos muito importantes para o programador, pois ele os vê, como a própria autora coloca, como estruturas de dados e funções (ver p. 49).

Entretanto, diferentes tipos de estruturas de dados e funções são passíveis de serem representadas em variáveis. No *Capítulo 5 - Ser e fazer em programas* a autora dá um exemplo em que o mesmo problema pode ser solucionado por softwares gerados por programas que, não obstante serem escritos na mesma linguagem (Java), são diferentes em sua estrutura. Ou seja, as variáveis num e noutro programa representam não somente estruturas diferentes, mas tipos de estruturas diferentes. Num programa, a classe '`Rectangle`' (uma variável) herda estruturas de dados e funções da classe '`Shape`' (outra variável), no outro programa a classe '`Rectangle`' implementa as interfaces '`Polygon`' e '`Area`' (ambas são variáveis também): nesse outro programa o programador não vê o conceito de '`Shape`'.

Essas diferentes implementações são possibilitadas pela linguagem de programação. Não houvesse o conceito de interface na linguagem Java, o segundo programa não poderia ter sido escrito; e o conceito é implementado pela palavra reservada '`implements`', que não é uma variável.

Depreende-se portanto que o poder expressivo das variáveis para o programador depende de outros signos nos programas que não as variáveis, e que estão ligados ao poder expressivo da linguagem de programação. Imaginamos que uma abordagem peirceana teria incluído esse tipo de semiose na análise semiótica da programação de computadores.

5.3 Lacunas na análise dos fenômenos identificados

Os fenômenos analisados pela autora são importantes, mas o trabalho poderia ter exposto outras perspectivas se a abordagem peirceana tivesse sido adotada. A visão dualista do signo limita a interpretação de fenômenos semióticos importantes. Uma lista provavelmente incompleta é mostrada a seguir.

5.3.1 Reflexividade e auto-recursão

O tema dos signos autorreferenciais é razoavelmente estudado na tradição peirceana (cf. NÖTH; SANTAELLA, 2014). É considerado autorreferencial o signo no qual o objeto e o representamen se confundem. Peirce diz-nos sobre isso:

> Mas para que qualquer coisa seja um Signo, ela deve 'representar', como se diz, outra coisa, chamada seu Objeto, embora a condição de que um Signo deva ser outro que não seu Objeto é talvez arbitrária, uma vez que, se insistirmos nisso, devemos ao menos fazer uma exceção para o caso em que um Signo é parte de um Signo. Assim, nada impede que o ator que atua como personagem num drama histórico ostente relíquias que deveriam estar apenas representadas, tais como o crucifixo que o Richelieu de Buwler ergue com tal efeito em sua rebeldia. Num mapa de uma ilha colocado sobre o chão dessa mesma ilha deve haver, em condições normais, alguma posição, algum ponto, assinalado ou não, que representa tal posição no mapa que é a mesma posição na ilha (CP 2.230, 1910).

Segundo Nöth e Santaella (2014), os princípios da autorreferencialidade sob a óptica peirceana encontraram um relevo mais geral na semiótica da cultura, nas estéticas do *Living Theater* e do *Happening*. Como se vê, a noção peirceana de signo autorreferencial vai muito além da frase que fala sobre si mesma, ou da função que tem definição recursiva.

A autora afirma que a autorreferência —no sentido de definição recursiva[2]— não é adequada para a computação (ver p. 57). Além disso, afirma que seres humanos e máquinas comportam-se de forma diferente ao interpretar uma autorreferência (ver *Capítulo 9 - Humanos estruturais versus computadores construtivos*), pois os humanos "têm a escolha de abandonar a tentativa de interpretação". Queremos aqui mostrar que, nesse caso, tanto homens como máquinas têm um comportamento semelhante no que toca ao raciocínio em si, residindo a diferença entre ambos os comportamentos num ponto fora dessa semiose. Ao dizer que seres humanos abandonam a tentativa de interpretação, a autora esquece que "abandonar a tentativa de interpretação" implica em não interpretar: exatamente o que ocorre com a máquina se lhe é dado interpretar (calcular) uma função reflexiva que não converge (ou seja, que não tem interpretação possível —não pode ser calculada). Ou seja, nem homens, nem máquinas interpretam funções reflexivas divergentes. A diferença é o que acontece com cada um depois de um certo número de tentativas de interpretação: o ser humano pode se dar conta de que a função é divergente —a partir de, por exemplo, um insight abdutivo, explicado em 4.2.4—, ou o ser humano pode simplesmente abandonar o cálculo após um certo número de iterações. Já a previsão exata do que uma máquina fará depende de fatores externos ao software: por exemplo, pode "congelar", ou interromper o processamento e emitir uma mensagem de erro. Exatamente como o humano: humanos, em algum momento, também interrompem o cálculo de uma função recursiva não convergente, e seu comportamento exato depende de fatores externos à função. Corroboramos assim a afirmação de Peirce que diz que as semioses humana e da máquina são idênticas quando são equivalentes a argumentos dedutivos, que pertencem à classe da secundidade. Uma função não-computável não é computável, nem para uma máquina, nem para um ser humano.

A diferença entre humanos e máquinas, que a autora atribui à capacidade de lidar com a reflexividade, na verdade reside no fato do ser humano, ao lidar com uma semiose da natureza da secundidade que não sabe se converge —como uma função auto-recursiva—, tem a alternativa de buscar outros caminhos através do raciocínio abdutivo, da natureza da primeiridade, e pode contar com o raciocínio indutivo, da natureza da terceiridade, para o auxiliar. Seres humanos podem por exemplo criar diagramas icônicos dessas deduções problemáticas e observá-los para tentar daí tirar ideias que não estão explícitas nas próprias deduções. A afirmação da autora, de que a autorreferência não é adequada à computação mas é tratada pelo ser humano parece-nos uma consequência de não ter utilizado uma abordagem peirceana genuína, que teria tomado as funções auto-recursivas como signo e estudado a sua semiose.

[2] Uma definição recursiva é aquela em que um objeto é definido em termos de si mesmo. O exemplo clássico é a definição de fatorial, designado pelo operador "!": se $n > 0$, temos que $n! = n \times (n-1)!$; e $0! = 1$.

Resta dizer que a única diferença real na semiose de um programa com definições recursivas comparada à semiose de um programa sem essas definições está na compreensão desses programas por um ser humano que os leia e procure entender o que faria o software derivado deles. O texto dos programas com funções recursivas têm menos elementos —a saber, as variáveis de controle das iterações não aparecem nas definições recursivas— e portanto exigem maior esforço para sua compreensão. Nos programas em que as variáveis de iteração recursiva são explícitas —ou seja, os programas que não têm definições recursivas— é muito mais fácil identificar as circunstâncias em que a recursão não é convergente e o software, consequentemente, não vai se comportar da maneira desejada. Portanto, não é o caso que a autorreferência não é adequada à computação; o caso é que ela não é adequada à compreensão dos programas por seres humanos. Talvez a única semiose que realmente não poderia deixar de ser analisada num texto que analisa a semiótica da programação no nível semântico da linguagem seja essa compreensão dos programas de computador por seres humanos.

5.3.2 A ambiguidade dos identificadores

A autora, no *Capítulo 6 - A afirmação '`x := x + 1`'*, avalia a expressão '`x := x + 1`' e afirma que nessa expressão os dois '`x`' não tem a mesma significação e por isso a variável '`x`' é um signo ambíguo; segundo ela o primeiro '`x`' corresponde a um endereço e o segundo a um valor, e aí reside a ambiguidade.

Entretanto a semiose da expressão se dá no contexto da linguagem de programação. Isso implica que, sendo o texto '`x := x + 1`' uma cadeia válida na linguagem de programação, há uma dedução necessária do software a partir do programa —a expressão dará origem, necessariamente, a um software. É o conhecimento da linguagem de programação que permite que um ser humano interprete corretamente a expressão '`x := x + 1`'. Portanto a expressão, se é uma cadeia válida da linguagem de programação, não é ambígua. E uma vez que a autora afirma que na linguagem de programação dada o símbolo '`:=`' corresponde ao operador de atribuição, podemos afirmar com certeza que a semiose da expressão não é ambígua, já que a autora nos informou como interpretá-la. A pergunta é: onde está a alegada ambiguidade?

A ambiguidade somente existe se tiramos a expressão de seu contexto e a lemos como equação matemática, assumindo que o operador de atribuição '`:=`' corresponde ao sinal de igualdade ('=') —equivale a tratar o símbolo '`:=`' como ícone, de modo que a semiose fica semelhante à de uma onomatopeia. Nessa caso teríamos uma expressão matemática afirmando que um número é igual a seu sucessor, o que é uma impossibilidade aritmética; e isso por si só já indicaria que, se a expressão estivesse correta, então deveria ser interpretada de outra maneira e não como uma equação matemática. E essa outra

maneira implicaria em afirmar que o primeiro '`x`' tem significação diferente do segundo, embora ambos sejam "a mesma coisa", já que são representados da mesma forma (a saber, '`x`') —donde a ambiguidade.

Ou seja: a autora prefere pensar na significação do símbolo '`x`' antes de pensar na significação da expressão que o contém. A expressão em si não é tratada como signo em nenhum momento. E a partir dessa visão passa a especular de maneira cartesiana a respeito dos conceitos de conotação e denotação, estendendo sua aplicação —aliás, de um modo difícil de corroborar à luz da teoria peirceana pura— aos conceitos de ícone, índice e símbolo de Peirce.

5.3.3 Semiose

No *Capítulo 9 - Humanos estruturais versus computadores construtivos* a autora apresenta o ponto culminante de sua interpretação do fenômeno da semiose, que parece seguir a linha saussureana. Nele, mostra um diagrama (reproduzido aqui na Figura 13, p. 71) que representaria as relações entre signos num modelo que chamou de estrutural, no qual o processo de semiose ocorre de alguma forma numa rede de signos inter-relacionados, e o modelo que chamou de construtivo, com signos inter-relacionados de uma maneira diferente. Assim, para ela, o processo de semiose se espalha numa rede de signos, com cada significado aludindo a outros signos, o que, de uma maneira desordenada como nos sistemas estruturais, leva a uma certa indeterminação nos significados, indeterminação que não existe nos sistemas construtivos, que são ordenados. É uma justificativa interessante e pode abrir caminhos para a análise da semiose com base nos relacionamentos entre os diversos signos de um sistema, análise essa que poderá prescindir do conceito peirceano de semiose.

A indeterminação do significado, que a autora justifica pela topologia da rede estrutural de signos, é considerada intrínseca à linguagem natural dentro do pensamento peirceano, como descrito em 4.3.3. Assim, numa abordagem peirceana, o alvo da análise não é a topologia da rede de signos. Isso pode trazer visões novas uma vez que, embora a topologia construtiva da rede de signos num programa de computador possa ser conhecida, a topologia das redes estruturais é uma incógnita, impossibilitando o estudo da indeterminação por esta via. Por outro lado, a compreensão da indeterminação intrínseca das linguagens naturais pode lançar luz a respeito das linguagens de computador e sua capacidade expressiva.

5.4 Desdobramentos possíveis

Não pretendemos aqui fazer a análise da programação de computadores à luz da semiótica de Peirce. Mas acreditamos possível delinear uma moldura sob a qual tal análise poderia ocorrer, apontando os caminhos que poderia tomar e especulando, com alguma liberdade, a respeito das possíveis consequências.

5.4.1 Questões centrais

Analisando o cap. 2 parece-nos que a programação de computadores lida com muitos fenômenos semióticos que pertencem à categoria peirceana da secundidade, a começar pelo próprio computador, que é uma máquina muito bem definida. De modo geral os fenômenos que pertencem à categoria da secundidade são estudados com muito sucesso pelas ciências: a Física, a Química e a Astronomia, por exemplo, criam modelos matemáticos muito exatos que permitem lidar com esses fenômenos quando encontrados no mundo natural. O próprio computador moderno (o equipamento em si) é um exemplo desse sucesso. O mesmo não ocorre com a ciência da computação, que tem falhado em apresentar uma solução para a eficiente criação de softwares (ver 2.5). Uma razão para isso pode estar na natureza da secundidade envolvida nessa atividade. Os elementos não físicos envolvidos na programação de computadores —máquina de Turing, computação, algoritmo, programa, linguagem de programação— também tem uma forte característica de secundidade na medida em que neles o raciocínio dedutivo é bastante presente, embora não se refiram a objetos existentes. Peirce nos deu (ver 4.3.1) um excelente caminho para iniciar o estudo das semioses que levam à formação das deduções. Uma questão central seria, então, justamente essa: as características do raciocínio diagramático e sua aplicação na formação de deduções, podendo incluir aí as próprias deduções matemáticas, mas certamente incluindo a criação de programas de computador.

Um estudo do raciocínio diagramático e sua aplicação às deduções envolvidas na programação de computadores e eventualmente na própria matemática poderia levar também a uma melhor compreensão de certos resultados matemáticos: se a semiótica de Peirce está correta, pode-se associar, como vimos (ver 5.2.2), os limites demonstrados no artigo de Turing (1936) ao fato de procedimentos computáveis não poderem lidar com fenômenos de primeiridade presentes no raciocínio diagramático. Caberia especular se essas considerações não se aplicam também a outros resultados semelhantes, como a prova de Gödel (1992, primeira publicação em 1931), ou ainda determinar a importância do raciocínio diagramático em procedimentos matemáticos como as demonstrações por absurdo, ou ainda lançar uma luz sobre os próprios fundamentos da matemática como são entendidos hoje (cf. COSTA, 1992).

5.4.2 Expressividade das linguagens de programação e o contexto da programação de computadores

Os programas de computador como são entendidos hoje, por serem necessariamente capazes de dar origem, de forma dedutível, a softwares de comportamento previsível, têm limitada capacidade de expressar os diagramas icônicos que deram origem às suas especificações. Uma consequência direta dessa limitação é que há dois aspectos expressivos a serem supridos: a expressão do software que será derivado do programa, e a expressão do diagrama que explicita a função pretendida pelo programa. Ambas não podem ocorrer no mesmo programa a não ser no caso em que a função pretendida é, ela mesma, da natureza da secundidade —caso em que o programa normalmente é trivial. Pode-se especular, inclusive, se a evolução histórica das linguagens de programação não se constituem de tentativas intuitivas para incorporar esse elemento icônico aos programas, na forma de abstrações e metáforas como "objetos" e "interfaces".

De qualquer modo, uma questão importante a abordar nesse caso é como dar conta dessas duas necessidades expressivas. Hoje em dia os processos de construção de software tentam dar conta dessas duas necessidade sem compreender inteiramente sua dupla natureza. Então, em algum ponto do processo de construção de software existe a observação de um diagrama icônico, observação esta que resulta em algo que ou poderá ser utilizado para deduzir o programa, ou é o próprio programa. Peirce, consciente da importância dos raciocínios diagramáticos, trabalhou na criação de seus Grafos Existenciais: será que estes não teriam utilidade no processo de especificação e construção de software?

Uma outra questão a lidar tem a ver com a vagueza intrínseca das linguagens naturais (ver 4.3.3). Uma vez que as linguagens de programação não podem se dar ao luxo da vagueza, é impossível que programas sejam especificados sobre bases vagas: todo processo de construção de programas implica em eliminar a vagueza, já que esta, se transmitida ao software, o leva a assumir comportamentos imprevisíveis. Mas será que isso significa que não se pode construir softwares sobre bases vagas? É fácil responder que não, uma vez que softwares derivam de programas que têm de ser precisos. Mas pode haver um caminho, na diferença entre softwares e programas: softwares são objetos existentes; programas não são necessariamente existentes (ver definições do cap. 2). E os seres humanos parecem ser capazes de, mesmo utilizando uma linguagem vaga, lidar com objetos existentes, e sobretudo construir novos objetos a partir de outros existentes, mesmo tendo um conhecimento apenas vago desses objetos —tanto dos já existentes quanto dos construídos. A vagueza na linguagem e no conhecimento não atrapalha um uso intuitivo do mundo que nos cerca. Ao contrário, ao reconhecermos nossa ignorância somos capazes superá-la, nem que seja pela tentativa e erro. Em nosso auxílio nessa empreitada estão diversos fatores, entre eles a regularidade das leis da natureza. A pergunta é: será que não

há uma maneira de construir softwares que se prestem a esse tipo de manipulação? Não seriam softwares derivados de programas, mas softwares derivados de outros softwares, de uma forma semelhante à que já ocorre em *games* (cf. MOJANG AB, 2015). Não se poderia estender esse tipo de manipulação, que incorpora a vagueza, à produção de qualquer tipo de software, construído com partes não-vagas de software?

5.4.3 Ambientes e comunidades de programação

Não obstante o descasamento entre o programa e o diagrama icônico de sua função pretendida observa-se situações que superam esse descasamento de uma maneira pública passível de investigação. As comunidades de desenvolvimento de software aberto, entre elas as comunidades dedicadas à criação de ambientes de programação, têm mecanismos que permitem que qualquer interessado seja capaz de atuar sobre os softwares e programas que criam e, portanto, suprem a dupla necessidade expressiva dos programas.

Uma abordagem semiótica peirceana da atividade de programação pode se valer desse acervo de desenvolvimento de sistemas, que é público, gratuito e disponível via internet, para investigar as necessidades expressivas que suprem e como o fazem. Um ramo interessante nessa investigação é o crescente número de ambientes de programação: muito mais do que linguagens, esses ambientes disponibilizam a quem os utiliza um modo completo de atuação que permite a rápida construção de softwares a partir de soluções já prontas, bastando ao usuário-programador "preencher" as partes faltantes com funcionalidades desejadas para seu próprio software.

5.4.4 Como lidar com a secundidade incorpórea

Um corolário possível dessa investigação talvez seja a possibilidade de encontrar um modo de utilizar o corpo de conhecimentos acumulado na investigação do mundo físico aos fenômenos semióticos incorpóreos. O esforço de Peirce no sentido de criar um edifício filosófico abrangente capaz de dar conta de todo o conhecimento científico poderia então ser melhor aproveitado, em um uso mais abrangente do que este que lhe estamos dando no presente trabalho.

6 Conclusão

A proposta do presente trabalho era estudar um caso de utilização da semiótica peirceana na análise da programação de computadores. Esperávamos, através da compreensão do que já foi feito, conseguir apontar caminhos possíveis para essa utilização. Isso está declarado no Capítulo 1: *Introdução*. Definições iniciais foram feitas no Capítulo 2: *Contexto*; o livro "*Semiotics of Programming*", de Tanaka-Ishii (2010) foi apresentado no Capítulo 3: *"Semiotics of Programming"*. Conceitos da semiótica peirceana apresentados no Capítulo 4: *Semiótica peirceana* foram utilizados na análise apresentada no Capítulo 5: *Semiótica de Peirce em "Semiotics of Programming"*.

Tanaka-Ishii apresentou importantes questões concernentes à ciência da computação, mas como observou Nadin (2011), utilizou os conceitos da semiótica em geral e da semiótica peirceana em particular de uma maneira superficial na sua análise. É preciso entretanto compreender as razões dessa superficialidade no uso dos conceitos de Peirce: eles exigem a compreensão de que se trata de uma outra epistemologia; exigem sabermos do quê se está falando quando se fala de semiótica em Peirce: de uma teoria da cognição. Esse entendimento principia por suas categorias, avança sobre os signos e suas classificações e desemboca no entendimento da semiose como vista por ele, trazendo de roldão todo o seu edifício filosófico. Não é tarefa que se leve a cabo sem esforço, e na qual é imprescindível o auxílio dos bons intérpretes de sua obra. Essas exigências não são evidentes à primeira vista, e esta é uma lacuna que oxalá este trabalho conseguirá ajudar a preencher.

Finalmente, embora a autora nos tenha frustrado na sua aplicação dos conceitos peirceanos, o estudo de seu livro não frustrou os objetivos do presente trabalho. Ao contrário, permitiu pavimentar um conjunto básico de conceitos tanto da computação quanto da semiótica peirceana que servirão, no futuro, para uma análise mais profunda da programação de computadores à luz de Peirce.

Referências

ABBOTT, R. J. Program design by informal english descriptions. *Communications of the ACM*, v. 26, n. 11, p. 882–894, nov. 1983. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/182.358441>. Acesso em: 9.4.2014.

ALGORITHM. In: ENCYCLOPAEDIA Britannica. Encyclopaedia Britannica, 2014. Disponível em: <http://global.britannica.com/EBchecked/topic/15174/algorithm>. Acesso em: 5.5.2014.

ANDERSEN, P. B. Computer semiotics. *Scandinavian Journal of Information systems*, v. 4, n. 1, p. 3–30 (article 1), 1992. Disponível em: <http://aisel.aisnet.org/sjis/vol4/iss1/1>. Acesso em: 13.9.2014.

_____. The force dynamics of interactive systems: toward a computer semiotics. *Semiotica*, v. 103, n. 1/2, p. 5–45, 1995.

BOOCH, G. et al. *Object-Oriented Analysis and Design with Applications*. 3. ed. Upper Saddle River, NJ: Addison-Wesley, 2007.

BRIER, S. *Cybersemiotics: Why Information is not Enough*. Toronto: University of Toronto Press, 2008.

BROOKS JR., F. P. No silver bullet – essence and accidents of software engineering. In: KUGLER, H. J. (Ed.). *Information processing 86*. Amsterdam: Elsevier, 1986. v. 20, n. 1. Disponível em: <http://www.cs.nott.ac.uk/~cah/G51ISS/Documents/NoSilverBullet.html>.

_____. *O mítico homem-mês, edição de 20o aniversário*. Tradução de Cesar Brod. Rio de Janeiro: Campus-Elsevier, 2009.

BRYANT, A. It's engineering Jim . . . but not as we know it: Software engineering – solution to the software crisis, or part of the problem? In: *Proceedings of the 22nd International Conference on Software Engineering*. New York, NY: ACM, 2000. (ICSE '00), p. 78–87. ISBN 1-58113-206-9. Disponível em: <http://doi.acm.org/10.1145/337180.337191>.

BUXTON, J. N.; RANDELL, B. (Eds.). *Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee*. Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO: [s.n.], 1970. Disponível em: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF>. Acesso em: 10.3.2014.

CARNIELLI, W. A.; EPSTEIN, R. L. *Computabilidade, funções computáveis, lógica e os fundamentos da matemática*. São Paulo: Editora UNESP, 2006.

CHURCH, A. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, v. 58, n. 2, p. 345–363, Abril 1936. Disponível em: <http://www.jstor.org/stable/2371045>. Acesso em: 21.6.2014.

COLUMBIA UNIVERSITY DEPARTMENT OF SURGERY. *A Brief History of Heart Transplantation*. 2014. Disponível em: <http://hearttransplant.com/history.html>. Acesso em: 5.3.2014.

COMPUTADOR. In: WIKIPEDIA. Wikimedia, 2014. Disponível em: <http://pt.wikipedia.org/wiki/Computador>. Acesso em: 18.6.2014.

COMPUTER. In: WIKIPEDIA. Wikimedia, 2014. Disponível em: <http://en.wikipedia.org/wiki/Computer>. Acesso em: 18.6.2014.

COSTA, N. C. A. da. *Introdução aos fundamentos da matemática*. 3. ed. São Paulo: Hucitec, 1992.

DESCARTES, R. *Discurso do método*. Tradução de Paulo Neves. Porto Alegre: L&PM, 2004. Introdução de Denis Lerrer Rosenfield.

_____. *The Principles of Philosophy, part I*. Translated by John Veitch. 2014. Disponível em: <http://www.gutenberg.org/ebooks/4391>. Acesso em: 23.5.2014.

DIJKSTRA, E. W. The humble programmer. *Communications of the ACM*, v. 15, n. 10, p. 859–866, 1972. Disponível em: <http://cs-exhibitions.uni-klu.ac.at/fileadmin/template/documents/text/p859-dijkstra.pdf>. Acesso em: 7.3.2014.

EBERBACH, E.; GOLDIN, D.; WEGNER, P. Turing's ideas and models of computation. In: TEUSCHER, C. (Ed.). *Alan Turing: Life and Legacy of a Great Thinker*. Berlin: Springer, 2004. p. 159–194. Disponível em: <http://www.cse.uconn.edu/~dqg/papers/turing04.pdf>. Acesso em: 5.5.2014.

ECO, U. *A Theory of Semiotics*. Bloomington, IN: Indiana University Press, 1979.

FANT, K. M. A critical review of the notion of algorithm in computer science. In: *Proceedings of the 1993 ACM Conference on Computer Science*. New York, NY: ACM, 1993. (CSC '93), p. 1–6. Disponível em: <http://staffweb.worc.ac.uk/DrC/Courses%202006-7/Comp%204070/Reading%20Materials/FAnt.pdf>. Acesso em: 19.4.2014.

FETZER, J. H. *Computers and Cognition: Why Minds are not Machines*. Dordrecht: Kluwer, 2001.

GAZENDAM, H. W. M. Organizational semiotics: a state of the art report. *Semiotix*, v. 1, n. 1, March 2004. Disponível em: <http://semioticon.com/sx-old-issues/semiotix1/sem-1-05.html>. Acesso em: 23.12.2014.

GÖDEL, K. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. Translated by B. Meltzer, Introduction by R. B. Braithwaite. New York, NY: Dover, 1992.

GUDWIN, R.; QUEIROZ, J. (Eds.). *Semiotics and Intelligent Systems Development*. Hershey, PA: Idea Group, 2007.

HEART TRANSPLANTATION. In: WIKIPEDIA. Wikimedia, 2014. Disponível em: <http://en.wikipedia.org/wiki/Heart_transplantation>. Acesso em: 5.3.2014.

HOARE, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM*, v. 12, n. 10, p. 576–583, October 1969.

HODGES, A. Alan Turing. In: ZALTA, E. N. (Ed.). *Stanford Encyclopedia of Philosophy*. Winter 2013. [s.n.], 2013. Disponível em: <http://plato.stanford.edu/archives/win2013/entries/turing/>. Acesso em: 21.6.2014.

JOHNSON, P.; EKSTEDT, M.; JACOBSON, I. Where's the theory for software engineering? *Software, IEEE*, v. 29, n. 5, p. 96–96, September 2012. Disponível em: <http://www.researchgate.net/publication/259079647_Where's_the_Theory_for_Software_Engineering/file/3deec529eb534b107e.pdf>. Acesso em: 9.4.2014.

JUNGK, I. *As dez classes possíveis de signo*. 2013. Comunicação pessoal.

KNUTH, D. E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 1. ed. Reading, MA: Addison-Wesley, 1968.

_____. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3. ed. Upper Saddle River, NJ: Addison-Wesley, 1997.

KOZEN, D. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, v. 1, n. 1, p. 60–76, jul. 2000. ISSN 1529-3785. Disponível em: <http://www.cs.cornell.edu/~kozen/papers/Hoare.pdf>. Acesso em: 19.5.2014.

LIU, K. *Semiotics in Information Systems Engineering*. Cambridge, MA: Cambridge University Press, 2000.

MARKOV, A. A. *Theory of Algorithms*. Translated by J. J. Schorr-Kon. Jerusalém: Keter Press, 1971.

MOJANG AB. *Minecraft*. 2015. Disponível em: <minecraft.net>. Acesso em: 11.01.2015.

MORENO, A. M. Object-oriented analysis from textual specifications. In: *Proceedings of 9 th International Conference on Software Engineeringand Knowledge Engineering (SEKE 97)*. [s.n.], 1997. Disponível em: <http://grise.upm.es/miembros/ana/docs/seke97.pdf>. Acesso em: 9.4.2014.

NADIN, M. Processos semióticos e de informação: a semiótica da computação. *TECCOGS - Revista Digital de Tecnologias Cognitivas*, v. 5, p. 89–121, 2011.

NAUR, P.; RANDELL, B. (Eds.). *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*. Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO: [s.n.], 1969. Disponível em: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>. Acesso em: 8.3.2014.

NÖTH, W. *Handbook of Semiotics*. Bloomington, IN: Indiana University Press, 1990.

_____. Máquinas semióticas. In: QUEIROZ, J.; LOULA, A.; GUDWIN, R. (Eds.). *Computação, Cognição, Semiose*. Salvador: EDUFBA, 2007. cap. 6, p. 159–183.

_____. Charles S. Peirce's theory of information: A theory of the growth of symbols and of knowledge. *Cybernetics & Human Knowing*, v. 19, n. 1-2, p. 99–123, 2012.

_____. *Instrumentality and Semiotic Agency of Signs, Tools and Intelligent Machines*. 2014. Manuscrito.

NÖTH, W.; GURICK, A. A teoria da informação de Charles S. Peirce. *TECCOGS - Revista Digital de Tecnologias Cognitivas*, v. 5, p. 4–29, 2011.

NÖTH, W.; SANTAELLA, L. Meanings and the vagueness of their embodiements. In: THELLEFSEN, T.; SØRENSEN, B.; COBLEY, P. (Eds.). *From First to Third via Cybersemiotics - A Festschrift Honoring Professor Søren Brier on the Occasion of his 60th Birthday*. Copenhagen: SL Forlagene, 2011. p. 247–282.

_____. *Introdução à semiótica*. [S.l.: s.n.], 2014. No prelo.

PEIRCE, C. S. Sign. In: BERGMAN, M.; PAAVOLA, S. (Eds.). *The Commens Dictionary: Peirce's Terms in His Own Words*. New Edition. Disponível em: <http://www.commens.org/dictionary/entry/quote-letters-lady-welby-43>. Acesso em: 6.11.2014.

_____. On a new list of categories. *Proceedings of the American Academy of Arts and Sciences*, v. 7, n. 1868, p. 287–298, 1868.

_____. Logical machines. *American Journal of Psychology*, v. 1, n. 1, p. 165–170, 1887. Disponível em: <http://history-computer.com/Library/Peirce.pdf>. Acesso em: 19.3.2014.

_____. *The Collected Papers of Charles Sanders Peirce*. Cambridge, MA: Harvard University Press, 1931–1958.

_____. *Semiótica*. Tradução de José Teixeira Coelho Neto. 2. ed. São Paulo: Perspectiva, 1995.

RAMOS, M. V. M.; NETO, J. J.; VEGA, Í. S. *Linguagens formais: teoria, modelagem e implementação*. Porto Alegre: Bookman, 2009.

RANSDELL, J. *Charles Peirce: the Idea of Representation*. [S.l.: s.n.], 1966. Tese de doutoramento inédita.

ROBINSON, H. Dualism. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2012. [s.n.], 2012. Disponível em: <http://plato.stanford.edu/archives/win2012/entries/dualism/>. Acesso em: 17.3.2014.

ROJAS, R. *A Tutorial Introduction to the Lambda Calculus*. 1997. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.4173>. Acesso em: 25.10.2014.

SANTAELLA, L. *A percepção: uma teoria semiótica*. 2. ed. São Paulo: Experimento, 1998.

_____. Methodeutics, the liveliest branch of semiotics. *Semiotica*, v. 124, n. 3/4, p. 377–395, 1999.

_____. *A teoria geral dos signos*. São Paulo: Pioneira, 2000.

_____. *Semiótica aplicada*. São Paulo: Thomson, 2002.

_____. *O método anticartesiano de C. S. Peirce*. São Paulo: Editora UNESP, 2004.

_____. *Percepção: fenomenologia, ecologia, semiótica*. São Paulo: Cengage Learning, 2012.

SANTOS, L. H. L. dos. A essência da proposição e a essência do mundo. In: SANTOS, L. H. L. dos (Ed.). *Tractatus Logico-Philosophicus, com introdução de Bertrand Russell, tradução, apresentação e ensaio introdutório de Luiz Henrique Lopes dos Santos*. São Paulo: EDUSP, 2008. p. 11–112.

SEBESTA, R. W. *Concepts of Programming Languages*. 10a. ed. Upper Saddle River, NJ: Pearson, 2012.

SKAGESTAD, P. The mind's machines: the Turing machine, the Memex, and the personal computer. *Semiotica*, v. 111, n. 3/4, p. 217—243, 1996.

SOUZA, C. S. de. *The Semiotic Engineering of Human-Computer Interaction*. Cambridge, MA: THe MIT Press, 2005.

STAMPER, R. et al. Understanding the roles of signs and norms in organisations. *Journal of Behaviour & Information Technology*, v. 19, n. 1, p. 15–27, 2000.

STAMPER, R. K. *Information in Business and Administrative Systems*. New York, NY: John Wiley & Sons, 1973.

STANDISH GROUP. *The CHAOS report (1994)*. 1994. Disponível em: <http://www.standishgroup.com/sample_research/showfile.php?File=chaos_report_1994.pdf>. Acesso em: 5.3.2014.

_____. *CHAOS Manifesto 2010*. 2010. Disponível em: <http://www.standishgroup.com/sample_research/showfile.php?File=CHAOSManifesto2010.pdf>. Acesso em: 5.3.2014.

_____. *CHAOS Manifesto 2013*. 2013. Disponível em: <http://versionone.com/assets/img/files/ChaosManifesto2013.pdf>. Acesso em: 7.3.2014.

STEINER, P. C.S. Peirce and artificial intelligence: historical heritage and (new) theoretical stakes. In: MÜLLER, V. C. (Ed.). *Philosophy and Artificial Intelligence*. Berlim: Springer, 2013. p. 265–276.

TANAKA-ISHII, K. *Semiotics of Programming*. Cambridge, MA: Cambridge University Press, 2010.

TELES, V. M. *Um estudo de caso da adoção de práticas e valores do Extreme Programming*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro - Núcleo de Computação Eletrônica, Rio de Janeiro, 2005. Disponível em: <http://www.improveit.com.br/xp/dissertacaoXP.pdf>. Acesso em: 7.3.2014.

TIERCELIN, C. Vagueness and the ontology of art. *Cognitio*, v. 6, n. 2, p. 221–253, 2005. Disponível em: <http://revistas.pucsp.br/index.php/cognitiofilosofia/article/view/13607>. Acesso em: 18.11.2014.

TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, v. 2, n. 42, p. 230–265, 1936. Disponível em: <http://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf>.

_____. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, s2-45, n. 1, p. 161–228, 1939.

_____. Lecture to the London Mathematical Society on 20 February 1947. In: *A. M. Turing's ACE Report of 1946 and Other Papers*. MIT Press, 1986. Disponível em: <http://www.vordenker.de/downloads/turing-vorlesung.pdf>. Acesso em: 23.6.2014.

VEGA, Í. S. *Conceitos Fundamentais de Modelagem de Software*. 2014. 11 fev. 2014, 10 jun. 2014. Notas de Aula.

VON NEUMANN ARCHITECTURE. In: WIKIPEDIA. Wikimedia, 2015. Disponível em: <http://en.wikipedia.org/wiki/Von_Neumann_architecture>. Acesso em: 19.1.2015.

VON NEUMANN, J. First draft of a report on the EDVAC. *Annals of the History of Computing, IEEE*, v. 15, p. 27–75, 1993. Disponível em: <http://www.virtualtravelog.net/wp/wp-content/media/2003-08-TheFirstDraft.pdf>. Acesso em: 6.6.2014.

WITTGENSTEIN, L. *Tratctatus logico-philosophicus*. Introdução de Bertrand Russell, tradução, apresentação e ensaio introdutório de Luiz Henrique Lopes dos Santos. 3a. ed. São Paulo: Edusp, 2001.

ZEMANEK, H. Semiotics and programming languages. *Communications of the ACM*, v. 9, n. 3, p. 139–143, 1966.