UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Caique Rodrigues Marques

APRESENTAÇÃO DA DEMONSTRAÇÃO DA EQUIVALÊNCIA ENTRE COMPUTABILIDADE E LAMBDA-DEFINIBILIDADE: UM ESTUDO DO TRABALHO DE ALAN TURING

Florianópolis

2019

Caique Rodrigues Marques

APRESENTAÇÃO DA DEMONSTRAÇÃO DA EQUIVALÊNCIA ENTRE COMPUTABILIDADE E LAMBDA-DEFINIBILIDADE: UM ESTUDO DO TRABALHO DE ALAN TURING

Trabalho de Conclusão de Curso submetido ao Programa de Ciências da Computação para a obtenção do Grau de Bacharelado em Ciências da Computação.
Orientadora: Profa. Dra. Jerusa Marchi

Florianópolis

2019

Caique Rodrigues Marques

APRESENTAÇÃO DA DEMONSTRAÇÃO DA EQUIVALÊNCIA ENTRE COMPUTABILIDADE E LAMBDA-DEFINIBILIDADE: UM ESTUDO DO TRABALHO DE ALAN TURING

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharelado em Ciências da Computação", e aprovado em sua forma final pelo Programa de Ciências da Computação.

Florianópolis, 14 de julho 2019.

Prof. José Francisco Danilo de Guadalupe Correa Fletes
Coordenador do Curso

Banca Examinadora:

Profa. Dra. Jerusa Marchi
Orientadora

Prof. Dr. Maicon Rafael Zatelli
Universidade Federal de Santa Catarina

Prof. Dr. Ricardo Azambuja Silveira
Universidade Federal de Santa Catarina

Este trabalho é dedicado à memória da minha avó Quitéria Batina Rodrigues.

AGRADECIMENTOS

Agradecimentos a meus pais e irmãos pelo eterno apoio.

Também agradeço aos colegas de curso, pelos ótimos momentos e por estarem perto durante toda a graduação.

Agradecimentos especiais aos membros do L3C, pelas experiências e aprendizado valiosos e eventuais conversas improdutivas.

Meus sinceros agradecimentos aos amigos de longa data, Matheus e Luiz.

Não podendo deixar de mencionar, agradeço ao Tarcísio, ao Charles e à Micheline pelo essencial suporte que me deram durante momentos mais necessários.

Agradeço à professora Jerusa pela orientação, apoio e dedicação.

We must know. We shall know
(David Hilbert, 1930)

RESUMO

Os conceitos que definem uma máquina de Turing, modelo teórico proposto por Alan Turing em 1937 (TURING, 1937b), servem de base para a computação em geral. Tais máquinas foram concebidas para solucionar um problema de decisão vigente na época: dado um algoritmo com um conjunto de axiomas e uma proposição matemática, ele é capaz de decidir que a proposição é ou não é provável pelos axiomas? (HILBERT; ACKERMANN, 1950) Para-relo ao trabalho de Turing, Alonzo Church propôs outra solução ao problema, com o uso do Cálculo Lambda (CHURCH, 1936).
Turing adicionou um apêndice no seu trabalho descrevendo um pouco sobre a equivalência dos dois modelos teóricos e, mais tarde, escreveu outra publicação que apresentaria com mais detalhes a prova (TURING, 1937a). Nesta publicação, Turing demonstra uma máquina que é capaz de computar Cálculo Lambda.
O objetivo deste trabalho é investigar a publicação de Turing sobre a equivalência de Cálculo Lambda e Máquinas de Turing, além de estudar os conceitos fundamentais e das motivações de ambos os modelos. É notório que os dois são capazes de computar a mesma classe de problemas, mas, a despeito de ambos os modelos serem vistos ao longo do curso, a demonstração da equivalência não é abordada.

Palavras-chave: máquinas de Turing, cálculo lambda, computabilidade.

ABSTRACT

The concepts that defines a Turing Machine, theoretical model proposed by Alan Turing in 1937 (TURING, 1937b), provides the basis for computing in general. These machines was conceived to solve a decision problem at the time: given an algorithm with a set of axioms and a mathematical setence, it is able to decide wheter or not the sentence is provable by the axioms? (HILBERT; ACKERMANN, 1950) Parallel to Turing's work, Alonzo Church proposed another solution to the problem, with the use of Lambda Calculus (CHURCH, 1936).
Turing added an appendix on his work describing about the equivalence of the two theoretical models and, later, he wrote another publication that would present with more details the proof (TURING, 1937a). In this publication, Turing demonstrates a machine that is capable of compute Lambda Calculus. This work intends to investigate the Turing's publication about the equivalence between Lambda Calculus and Turing Machines, in addition to studying the fundamental concepts and motivation of both models It is notorious that both models are capable to compute the same class of problems, but, despite of both models being seen along the course, the demonstration of equivalence isn't addressed.

Keywords: turing machines, lambda calculus, computability.

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

Um problema de decisão é uma função com uma saída: "sim" ou "não" (KOZEN, 1997), se o problema possui tal característica ele é dito *decidível*, caso contrário, é *indecidível*. A decibilidade de um problema define se ele é resolvível por uma máquina, sendo que a máquina possui recursos limitados e a ciência da computação é majoritariamente devotada a resolver problemas, é crucial saber tomar as medidas cabíveis para solucionar um problema: sendo *decidível*, saber a melhor forma de aproveitar os recursos a fim de resolvê-lo; sendo *indecidível*, tomar medidas para simplificá-lo ou alterá-lo a fim de torná-lo algo solucionável para uma máquina.

Na tentativa da formalização da matemática, David Hilbert propôs, em 1928, um desafio à comunidade matemática: ele pedia por um algoritmo que recebesse como entrada qualquer sentença válida em lógica de primeira ordem e que o algoritmo fosse capaz de retornar uma resposta do tipo "sim" ou "não". O desafio é chamado de *Entscheidungsproblem* ("problema de decisão" em alemão). A primeira solução ao problema foi apresentada por Alonzo Church (CHURCH, 1936), em 1936, apresentando que não existe tal algoritmo. Pouco depois e independente a Church, Alan Mathison Turing propôs outra solução também apresentando a impossibilidade da existência do algoritmo (TURING, 1937b). Antes da publicação de sua tese, portanto, Turing aprendeu sobre a solução de Church, o que o obrigou a adicionar um apêndice em sua publicação mostrando que ambas as abordagens são equivalentes (PETZOLD, 2008).

A proposta de Church ao desafio de Hilbert, chamada de Cálculo Lambda, consiste na construção de alguns lambda termos e realizando operações entre eles, seguindo uma série de regras, é possível produzir expressões lógicas. O Cálculo Lambda é percursor das linguagens de programação para computadores, mais notadamente, as de paradigma funcional como é o caso das linguagens Haskell e LISP (BARENDREGT, 1997). A proposta de Turing, nomeada de Máquina de Turing, é um modelo teórico de uma máquina com memória infinita e uma fita na qual os valores de entrada são lidos pela máquina. A Máquina de Turing é o modelo mais preciso de uma máquina de propósito geral, assim, uma Máquina de Turing é capaz de realizar tudo o que uma máquina de propósito geral é capaz de fazer (SIPSER, 2013).

É possível notar que ambas as abordagens computam a mesma classe de problemas, assim, o objetivo deste trabalho é estudar a publicação de Turing sobre a equivalência dos modelos Máquinas de Turing e Cálculo Lambda, examinando cada uma das abordagens e analisando cada passo da prova, elucidando cada detalhe.

1.1 OBJETIVOS

Objetivo geral: Investigar os trabalhos de Alan Turing e Alonzo Church, apresentando a demonstação da equivalência entre seus modelos.

1.1.1 Objetivos específicos

1. Estudar os modelos de lambda definibilidade de Church e computabilidade de Turing;
2. Apresentar as definições formais de Cálculo Lambda e de Máquinas de Turing;
3. Compreender e apresentar a demonstração de equivalência entre os modelos apresentada por Turing em seu trabalho de 1937 (TURING, 1937a).

1.2 ESTRUTURAÇÃO DO TEXTO

O capítulo 2 compreende a fundamentação teórica do trabalho contendo, na seção 2.1, o histórico, que contextualiza os eventos relacionados que levaram à criação da Máquinas de Turing e do Cálculo Lambda, partindo da crise da matemática do início do século XX, seguindo com o surgimento das escolas da filosofia da matemática, os problemas de Hilbert, o teorema da incompletude de Gödel, o *Entscheidungsproblem* e a tese de Church-Turing, finalizando o relato histórico do surgimento dos modelos teóricos; nas seções 2.2 e 2.3, há a definição formal dos modelos de Cálculo Lambda e Máquinas de Turing, respectivamente, apresentando os conceitos e definições formais de cada um, finalizando o capítulo, a seção 2.3.4 mostra sobre as *skeleton tables*.

O capítulo 3 detalha a relação e a equivalência entre Máquinas de Turing e Cálculo Lambda, apresentando o estudo do trabalho de Turing publicado em 1937 (TURING, 1937a). O capítulo percorre pelas seções da publicação original, começando com a introdução, seguindo com as definições do Cálculo $\lambda - K$ e das abreviações, a criação das tabelas de conversão de Cálculo $\lambda - K$ e, finalizando o capítulo, a computabilidade das funções $\lambda - K$.

O capítulo 4 apresenta as considerações finais deste trabalho, retomando alguns pontos levantados, observando os legados e apresenta os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 HISTÓRICO

No início do século XX, matemáticos passaram a se questionar sobre a veracidade dos fundamentos da área. Desde o surgimento da matemática, diversas ideias foram consideradas como verdade e muitas delas foram rigorosamente avaliadas, porém, os matemáticos questionavam se tudo o que até então era conhecido era, de fato, verdadeiro. Paradoxos, principalmente na teoria de conjuntos, e contradições foram os pontos que culminaram em dúvidas sobre o que poderia ser afetado e qual o impacto que deixaria (IRVINE; DEUTSCH, 2016) - a citar, tais como o paradoxo de Russell[1], o paradoxo de Cantor[2], etc. (FILHO, 2007).

Para ter a garantia que a matemática estaria bem estabelecida, os matemáticos mostrariam que os teoremas provados como verdadeiros poderiam ser derivados de outros teoremas provados como verdadeiros e assim sucessivamente até todos eles derivarem dos axiomas. Os axiomas são considerados verdadeiros sem uma prova, pois ela é evidente, portanto, a matemática deveria ser construída a partir dos axiomas. Surgiram as escolas da filosofia da matemática para discutir os fundamentos da matemática, dentre as três que se destacaram e se opunham entre si: a escola intuicionista, a escola logicista e a escola formalista (FILHO, 2007) (YANDELL, 2002).

As ideias defendidas pela escola logicista tiveram base na obra *Fundamentos da Aritmética*, publicada em 1884, pelo filósofo Friedrich Ludwig Gottlob Frege (1848-1925), no entanto, a obra permaneceu esquecida até o matemático Bertrand Russell (1872-1970) redescobrir suas ideias, de modo independente. Na época em que o livro foi publicado, a análise matemática estava passando por um processo de *arimetização*. Quaisquer definições e proposições ambíguas foram descartadas, chegando a um ponto onde a base de toda a análise matemática seriam os números naturais - assim, os conceitos de números inteiros, complexos, reais,..., foram rigorosamente definidos

[1] O paradoxo foi descoberto por Georg Cantor em 1899. Pelo teorema de Cantor, a cardinalidade de qualquer conjunto é menor do que a cardinalidade do conjunto formado por todos os seus subconjuntos. Sendo S um conjunto de todos os conjuntos, então os subconjuntos de S também são seus elementos. Portanto, a cardinalidade de S é maior do que a cardinalidade do conjunto de subconjuntos de S, o que contraria o teorema.

[2] O paradoxo de Russell, descoberto por Bertrand Russel entre maio e início de junho de 1901 (LINK, 2004) e publicado em 1903, é um paradoxo autorreferente. Sendo um conjunto S cujos elementos são conjuntos que não têm a si mesmos como elementos, S está em S? A resposta é sim e não, tendo assim um paradoxo (RUSSELL, 1903). Uma das soluções para o paradoxo, foi a teoria dos tipos, proposta por Russell.

através da aritmética[3] (COSTA, 2008).

A tese logiscista voltou a ganhar atenção com a publicação da obra *Principia Mathematica*, por Russell e Alfred North Whitehead (1861-1947), que foi publicada em três volumes em 1910, 1912 e 1913. A tese defendia que a matemática era redutível à lógica, assim, é composta por dois conceitos principais:

> 1) toda a ideia matemática pode ser definida por intermédio dos conceitos lógicos (por exemplo, classe ou conjunto, relação, implicação, etc.); 2) todo enunciado matemático verdadeiro pode ser demonstrado a partir de princípios lógicos, mediante raciocínios puramente lógicos [...] (COSTA, 2008)

Pode-se citar como exemplos de princípios lógicos as três leis tradicionais do pensamento: a lei da identidade, que descreve que cada sentença é igual a si mesma, simbolicamente representando, por exemplo, descreve que $1 = 1$; a lei da não contradição, que descreve que duas ou mais sentenças contraditórias não podem ser, ao mesmo tempo, verdadeiras; e a lei do terceiro excluído, que descreve que para qualquer proposição, ou ela é verdadeira, ou ela é falsa (RUSSELL, 1912). A escola apresentou progressos ao mostrar a união entre a matemática e a lógica (FILHO, 2007).

A escola intuicionista, fundada pelo geômetra Luitzen Egbertus Jan Brouwer (1881-1966), tem bases no *finitismo*, tese de Leopold Kronecker (1823-1891). Kronecker defendia que os números naturais eram a base da matemática e a partir deles deveriam ser construídas as teorias, portanto, partindo dos números naturais, tudo na matemática deveria ser intuitivo e resultado de um processo de construção realizado pelo matemático. Kronecker não aceitava a existência de objetos infinitos na matemática, por exemplo, segundo ele o conjunto dos números naturais {0, 1, 2, ...} é algo que foi construído - definindo um elemento inicial, o zero, e seguindo uma lei de formação, a soma por um, o conjunto é construído - ainda assim ele não analisava o conjunto como infinito, pois eventualmente o processo de construção iria *parar*, logo, apenas conjuntos finitos poderiam ser analisados como algo dado, porque foram efetivamente *construídos*. Kronecker viria a tentar construir diversas outras teorias para sustentar a tese do finitismo, mesmo que elas contradissessem outras teorias já estabelecidas (COSTA, 2008).

As ideias do finitismo acabaram contradizendo as noções básicas também definidas na lógica formal clássica que, segundo Kronecker, era imprópria para a matemática. Por exemplo, o princípio do terceiro excluído e a prova por contradição[4], segundo Kronecker, não são aplicáveis em todos os

[3] Dentre os responsáveis pela definição dos números estão Georg Cantor, Richard Dedekind e Karl Weierstrass

casos. Com os avanços providos pela teoria de conjuntos de Georg Cantor (1845-1918), teoria ao qual Kronecker opunha fortemente, a tese do finitismo foi deixada de lado pelos matemáticos durante o final do século XIX, apesar de alguns seguirem ideias derivadas desta (COSTA, 2008).

O intuicionismo, portanto, defendia que a matemática é proveniente de um processo de construção mental, sem o uso da linguagem matemática como um instrumento, pois para Brouwer a linguagem só deveria ser usada como uma tentativa de sugerir construções (YANDELL, 2002). Similar ao finitismo, no intuicionismo, todas as construções teriam como base a *intuição* e qualquer prova que fosse não-construtiva era rejeitada pelos intuicionistas. Brouwer defendia que a matemática era autossuficiente, assim, ela não teria bases na lógica (COSTA, 2008).

A escola formalista, fundada por David Hilbert (1862-1943), defendia que a matemática deveria seguir um método axiomático, que procedia da seguinte forma: há o estabelecimento de axiomas, ou "verdades óbvias", que servem como base para a construção dos conhecimentos matemáticos, tais axiomas são considerados verdadeiros, sem a necessidade de uma prova, assim toda proposição que viesse a ser apresentada, tomaria os axiomas como premissas e, numa sequência de argumentos considerados válidos, se chega a uma conclusão (COSTA, 2008). Idealmente, o método deveria conter quatro características únicas (PETZOLD, 2008):

- *Independência*: garante que os axiomas sejam únicos, ou seja, não é possível obter um axioma partindo de outros. Por muito tempo, matemáticos achavam que o quinto postulado[5] de Euclides fosse derivado dos quatro primeiros, porém, em 1868 o matemático Eugenio Beltrami (1835-1900) viria a provar que os cinco postulados são, de fato, independentes.

- *Consistência*: significa que partindo dos axiomas não é possível derivar dois teoremas que são contraditórios entre em si. Logo, se uma sentença é comprovada verdadeira por um método, não deveria ser possível comprovar que ela é falsa por outro método.

- *Completude*: implica que todas as sentenças verdadeiras são derivadas a partir dos axiomas, se uma sentença não for derivada dos axiomas, então o método é dito incompleto. Detalhe para que existe diferenças

[4]Uma prova por contradição começa assumindo a negação da proposição como verdadeira, e assim é apresentada uma série de passos que demonstra que a tal proposição leva a uma contradição. Um exemplo que ilustra o uso de uma prova por contradição é a demonstração de que $\sqrt{2}$ é irracional, explicada em (SINGH, 2014, Apêndice 2).

[5]Postulados são as "verdades incontestáveis"na geometria, enquanto axioma se aplica universalmente.

entre o que é *verdadeiro* e o que é *provável*: existem sentenças que são consideradas verdadeiras sem o conhecimento de uma prova, tais sentenças são chamadas de conjecturas. - apesar de que é possível ter uma ideia do porquê são verdadeiras, ainda não há nada que as justifique por completo. Por exemplo, a conjectura de Goldbach define que todo inteiro par maior que 2 pode ser escrito como uma soma de dois números primos. A conjectura é amplamente aceita e é possível verificar que ela se aplica até a um valor máximo[6], porém, ainda não existe uma prova que confirma que é válida para *todos* os inteiros.

- *Decibilidade*: permitiria encontrar a provabilidade de uma sentença matemática, ou seja, Hilbert estava querendo saber se existe algum modo de conseguir determinar se uma dada sentença é verdadeira ou falsa. Na aritmética estão os problemas de decisão mais conhecidos[7], por exemplo, para saber se um número a é divisível por b basta realizar a operação de divisão aritmética entre ambos, após um finito número de passos, se chega a um resultado tendo um resto e um quociente. Se o resto é igual a zero, então a é divisível por b, caso contrário, a não é divisível por b. Em tais casos existe um método, um algoritmo[8], para encontrar a solução, para decidir se a sentença (*a é divisível por b*) é verdadeira ou não. Outro problema aritmético envolve saber se é finita ou infinita a sequência de números primos ímpares que se sucedem, assim, a sequência 11-13, 17-19, 41-43, ... é infinita ou finita? Ainda não existe uma resposta para a pergunta, como também não se sabe como é o padrão dos números primos, assim, o problema é indecidível.

Em 1900, David Hilbert foi chamado para apresentar uma conferência no Segundo Congresso Internacional de Matemáticos, em Paris. Hilbert já tinha feito contribuições à álgebra, à teoria de números, à geometria e ainda estava fazendo alguns estudos em análise matemática (YANDELL, 2002). Para a conferência, Hilbert decidiu apresentar os problemas que, segundo ele, seriam pertinentes e importantes para o novo século XX, problemas que deveriam trazer as soluções para os questionamentos que então estavam sendo feitos à matemática. Devido ao tempo da conferência ele apenas apresen-

[6] Até então, foi confirmado que a conjectura permanece verdadeira para inteiros pares menores que 4×10^{18} (SILVA; HERZOG; PARDI, 2014).

[7] Exemplos como apresentados em (FILHO, 2007)

[8] O uso do termo "*algoritmo*"como é usado atualmente, ou seja, para indicar a um procedimento com uma sequência de passos finitos que para, começou com os livros de computação da década de 1960 (PETZOLD, 2008). A partir de 1950, o termo foi comumente associado ao algoritmo de Euclides, que é um método que retorna o máximo divisor comum de dois números. No entanto, o nome "algoritmo de Euclides"começou a ser mais usado no começo do século XX (KNUTH, 1997).

tou dez problemas, de uma lista de vinte e três - todos os problemas seriam publicados mais tarde[9].

2.1.1 Os Teoremas da incompletude de Gödel

Da lista dos vinte e três problemas citados por Hilbert, o segundo consistia em

> [...] provar que eles [os axiomas da aritmética] não são contraditórios, isto é, que um número finito de passos lógicos baseados neles nunca deve chegar a resultados contraditórios (HILBERT, 1902, tradução nossa).[10]

Assim, era necessário ter a garantia de que as bases que formam a aritmética estavam bem definifidas, porém, em 1931, Kurt Gödel (1906-1978) viria a provar que não seria possível garantir a *consistência* nos axiomas.

2.1.1.1 A aritmética de Peano

O matemático Giuseppe Peano (1858-1932) escreveu a obra *Arithmetices principia, nova methodo exposita* (HEIJENOORT, 1967), publicada em 1889, que resumia e simplificava o trabalho de Richard Dedekind (1831-1916), que havia proposto a axiomatização dos números naturais. Na obra, Peano descreve, em lógica de primeira ordem, as características dos números naturais em uma lista de axiomas[11], partindo destes axiomas, ele constrói os teoremas que definem as propriedades das operações aritméticas.

O primeiro teorema que Peano prova, sendo o central da aritmética, descreve o conceito de *sucessor*, ou seja, todo o número natural x tem um sucessor - detalhe que, neste ponto, ainda não é possível descrever o sucessor de x como $x+1$, pois os conceitos relacionados à adição ainda não haviam sido definidos. O conceito também tornou-se importante por dois motivos: o primeiro é que, pela natureza de sempre haver um "próximo número", isso estabelece a existência de infinitos números; o segundo é base para a construção das propriedades das operações elementares da aritmética como a adição,

[9] Originalmente, Hilbert tinha em mente um vigésimo quarto problema, porém, não foi divulgado na conferência e nem publicado com os outros vinte e três. O problema foi redescorberto em 2003 com a publicação do historiador Rüdiger Thiele, que descobriu sobre o vigésimo quarto problema nos estudos das notas pessoais de David Hilbert (THIELE, 2003).

[10] Tradução livre do seguinte trecho: "*to prove that they are not contraditory, that is, that a finite number of logical steps based upon them can never lead to a contradictory results*"

a subtração, a multiplicação, a divisão, etc. (FILHO, 2007).

2.1.1.2 Numeração de Gödel

Com a aritmética de Peano, Gödel tentaria provar que ela conteria quatro propriedades importantes: ela é construída por um número finito de passos[12]; ela é consistente; ela é completa; e seria suficientemente poderosa para descrever todas as proposições feitas com números naturais (CASTI, 1996). Uma função desenvolvida por Gödel o ajudaria a formalizar a prova, chamada de numeração de Gödel, com a função, seria possível ter um modelo formal "isolado", assim teria um ambiente em que pudesse avaliar as propriedades da aritmética.

Na função que Gödel propôs, ela é capaz de expressar através de uma linguagem, assim, a linguagem usada é a lógica simbólica apresentada por Russel e Whitehead na obra *Principia Mathematica*. Cada caractere do alfabeto lógico é associado a um número chamado de *número de Gödel*, como ilustra um pequeno exemplo na tabela 1, cada um dos números de Gödel da sequência é, então, associado como uma potência do respectivo número na sequência de números primos, o resultado da multiplicação de tal sequência é a saída resultante da função. No caso de variáveis, cada uma é associada a um número primo maior que 10.

Por exemplo[13], a seguinte expressão "*Existe um número x que é sucessor imediato do número y*" é transcrito em lógica símbólica como $(\exists)(x = \mathbf{s}y)$ e traduzindo cada caractere da expressão pelo respectivo número de Gödel temos a seguinte sequência: $\{8,4,11,9,8,11,5,7,13,9\}$. Por fim, cada elemento da sequência é a potência do respectivo número da sequência de primos, assim:

$$2^8 \times 3^4 \times 5^{11} \times 7^9 \times 11^8 \times 13^{11} \times 17^5 \times 19^7 \times 23^{13} \times 29^9.$$

A multiplicação resulta no número m. Com a função de Gödel, ele poderia garantir que cada expressão é única porque ela tem um número único associado, portanto, evita ambiguidades.

Tabela 1: Numeração de Gödel de caracteres lógicos.

[11] Alguns exemplos: os axiomas de Peano números 1, 2 e 3 descrevem, respectivamente, que o número 1 pertence ao conjunto dos números naturais; se um número a pertence ao conjunto dos números naturais, então a é igual a a; se a e b pertencem ao conjunto dos números naturais, então $(a = b)$ é igual a $(b = a)$.

[12] Se não fosse possível construir em um número finito de passos, então a aritmética não seria *decidível*.

[13] Exemplo baseado em (CASTI, 1996).

Sinal	Número de Gödel	Significado
~	1	Negação
$\vee$	2	Ou
$\supset$	3	Se... então
$\exists$	4	Existe
$=$	5	É igual
0	6	Zero
s	7	O sucessor imediato
(	8	Abre parênteses
)	9	Fecha parênteses
,	10	Virgula
x	11	Variável
y	13	Variável

2.1.1.3 Demonstração da incompletude

Os paradoxos lógicos, notadamente os paradoxos autorreferentes, causaram grande preocupação à comunidade matemática, sendo o exemplo mais notável o paradoxo do mentiroso. O paradoxo do mentiroso apresenta uma sentença dita por um locutor sobre ele mesmo estar mentindo, por exemplo, as frases "eu estou mentido" ou "todos os cretenses são mentirosos", esta sendo dita por um cretense (YANDELL, 2002). Se o locutor estiver mentindo, então ele está falando a verdade, o que implica que ele está mentindo.
Gödel usaria o paradoxo do mentiroso como uma ferramenta para a demonstração da incompletude. No entanto, Alfred Tarski (1901-1983) já havia mostrado antes que o conceito de verdade não é algo que poderia ser bem estabelecido em um sistema formal. Assim, Gödel fez uma modificação em que o paradoxo seria compatível com o seu sistema formal (CASTI, 1996).
Para maior especificação em seu sistema formal, Gödel escreveu quatro teoremas que definiriam a construção de funções recursivas e, baseando-se nestas, ele propôs quarenta e cinco funções. Duas funções propostas por Gödel possibilitaram a ele adaptar o paradoxo do mentiroso ao domínio de seu sistema formal:

- **Demonstrabilidade**: A função *D*, é definida como "*a sequência de fórmulas de número de Gödel y é uma demonstração para a fórmula de número de Gödel x*", simbolicamente, a sentença é definida como *yDx*. Disto, Gödel define a função demonstrabilidade, enunciada como "*a fórmula x é demonstrável se, e somente se, existe uma sequência de fórmulas q que define x*";

- **Substituição**: A função de substituição, definida como *subst*, permite que seja possível, em uma fórmula, substituir uma variável por um valor numérico.

Para a prova da incompletude, Gödel começou com a seguinte sentença:

esta sentença é falsa.

Levando em conta os estudos de Tarski, a sentença foi reformulada a fim de que seja compatível com o sistema formal proposto por Gödel, portanto, a versão modificada da sentença fica:

esta sentença não é demonstrável.

A sentença é autorreferente, assim, a "sentença" mencionada é ela mesma, portanto, ela pode ser descrita, em terceira pessoa, como:

a sentença de número de Gödel x não é demonstrável.

Com a numeração de Gödel foi possível transformar a sentença em um número único que a expressa, assim, seja g tal número e $F(x)$ a função descrita pela sentença. Se substituir a variável x pelo número de Gödel g na função $F(x)$, teremos $F(g)$ que significa:

a sentença [de número de Gödel] a sentença [de número de Gödel] x não é demonstrável não é demonstrável.

Portanto, no sistema formal isso implica que o resultado de $F(g)$ não é formalmente demonstrável, apesar de, em seguida, Gödel demonstrar que é verdadeira para todos os números naturais. Analogamente, a negação do resultado, que é falso, também não é formalmente demonstrável e como não é possível se concluir à sentença, isto também define que o resultado de $F(g)$ é indecidível. Assim, o sistema formal de Gödel que avalia a aritmética mostrou que esta é *incompleta*, pois existem proposições não demonstráveis (FILHO, 2007).
Publicados em 1931, os teoremas da incompletude de Gödel tiveram um grande impacto no programa de Hilbert, que achava que a matemática poderia ser contida num sistema formal (YANDELL, 2002).

2.1.2 *Entscheidungsproblem*

Em 1928, Wilhelm Ackermann (1896-1962) e David Hilbert publicaram a obra *Grundzüge der theoretischen Logik*, que compila as palestras de lógica matemática ministradas por Hilbert na Universidade de Göttingen. O livro

trouxe os formalismos que fomentariam a criação da lógica de primeira ordem (ou cálculo de predicados de primeira ordem) e discute sobre a completude e a decibilidade desta. Assim, das seções abordadas por Hilbert e Ackermann, eles apontaram:

> [...] lá que emerge a fundamental importância de determinar se uma dada fórmula de cálculo de predicados é ou não universalmente válida. [...] [A] validade universal de uma fórmula significa o mesmo que sua validade universal em cada domínio dos indivíduos. [...] Este fato não garante uma solução para o problema da validade universal, já que não temos um critério geral para a deducibilidade de uma fórmula (HILBERT; ACKERMANN, 1950, tradução nossa).[14]

O capítulo em que há a discussão da decibilidade é chamado *Entscheidungsproblem* ("problema de decisão" em alemão), Hilbert e Ackermann, portanto, procuravam saber se existe algum procedimento que seja capaz de decidir se uma fórmula em lógica de primera ordem é válida ou não, ou seja, se tal fórmula é ou não demonstrável.
No mesmo ano, no Congresso Internacional de Matemáticos, em Bolonha, em continuação à lista de problemas que Hilbert havia proposto no congresso de Paris em 1900, ele apresentou mais detalhes do *Entscheidungsproblem* (HODGES, 1983). A primeira abordagem apresentando a resposta negativa quanto à existência de um procedimento que fosse capaz de decidir sobre uma fórmula de lógica de primeira ordem, veio em 1936 por Alonzo Church (1903-1995), que introduziu o Cálculo Lambda. Pouco depois e independente, Alan Turing (1912-1954) apresentou outra abordagem ao problema usando, o que seriam chamadas mais tarde, de Máquinas de Turing.

2.1.3 A tese de Church-Turing

O enunciado do décimo problema[15] de Hilbert descreve:

> Dada uma equação diofantina com um número qualquer de variáveis desconhecidas e com coeficientes numéricos integrais racionais: *Criar um processo que possa ser determinado por um*

[14] Tradução livre do trecho: "[...] there emerges the fundamental importance of determining whether or not a given formula of the predicate calculus is universally valid. [...] [T]he universal validity of a formula means the same as its universal validity in every domain of individuals. [...] This fact does not yield a solution of the problem of universal validity, since we have no general criterion for the deducibility of a formula.".

[15] Para mais detalhes sobre a história do décimo problema, ver (YANDELL, 2002, '*Can't We Do This with a Computer?: The Tenth Problem*').

> *número finito de operações se a equação é resolvível em inteiros racionais* (HILBERT, 1902, tradução nossa).[16]

O problema consiste em, se possível, formular algum *processo* que consiga decidir se uma dada equação diofantina[17] que cumpra os requisitos mencionados é resolvível usando inteiros racionais. Tal processo deveria ser capaz de *decidir* se a dada equação diofantina é ou não resolvível, matematicamente, tal processo poderia ser uma função que resultaria em 1 caso a resposta seja "sim", a equação é resolvível, ou resultaria em 0 caso a resposta seja "não", a equação não é resolvível. Disto, os processos para resolução das equações diofantinas, que são funções, seriam subconjuntos das funções que são "efetivamente calculáveis" (YANDELL, 2002).

Nos anos 1920, os matemáticos notaram que é possível construir funções partindo de regras simples e repetindo-as recursivamente[18], disto, foi definido o conceito de funções recursivas primitivas, que são funções compostas usando composição e uma recursão primitiva. A construção das funções recursivas primitivas foi uma tentativa de conseguir, a partir delas, englobar todas as funções efetivamente calculáveis. No entanto, em 1928, Ackermann viria a apresentar uma função efetivamente calculável, mas que não é recursiva primitiva. O conceito de efetivamente calculável ainda estava em aberto.

Gödel foi um dos matemáticos que trabalhou com os conceitos de recursividade e, em 1931, recebeu uma carta de Jacques Herbrand (1908-1931) apresentando algumas propostas que ele tinha para a definição de efetivamente calculável, muitas das sugestões de Herbrand não resolviam o problema e Gödel não as considerou a primeira vista. (YANDELL, 2002) (HEIJENOORT, 1967).

Quando formulado em 1932 (CHURCH, 1932), o cálculo lambda tinha como objetivo desenvolver uma teoria geral de funções e, disto, fornecer as bases para a lógica e partes da matemática. No entanto, Stephen Cole Kleene (1909-1994) e John Barkeley Rosser (1907-1989) viriam a provar que o sistema de Church é inconsistente como base para a lógica, assim, apresentando a existência de um paradoxo no cálculo lambda, que levou o nome de paradoxo

[16] Tradução livre do trecho: "Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers*."

[17] O nome se refere ao matemático grego Diofanto de Alexandria. Uma equação diofantina é uma equação polinomial com uma ou mais variáveis e coeficientes, por exemplo, $10x^3 + 42x^2 - x + 17$.

[18] Na matemática, a recursão é um processo onde um elemento é construído a partir de elementos do mesmo tipo. Os números de Fibonacci são o exemplo mais comum de recursão, eles são uma sequência de números, iniciada por 0 e 1, onde o próximo da sequência é a soma dos anteriores, assim, $0, 1, 1, 2, 3, 5, 8, 13, 21, ...$

de Kleene-Rosser. Desistido do objetivo de formular as bases da matemática com o cálculo lambda, assim, Church estava focando na parte livre de inconsistências e, ainda, estava tendo mais sucesso com o objetivo do desenvolvimento geral de funções, culminando com a tentativa da definição do conceito de efetivamente calculável aplicado em cálculo lambda, que é chamado de λ-definível (BARENDREGT, 1984).

Church conversou com Gödel sobre o trabalho que estava tendo em relação ao conceito de λ-definível, no entanto, Gödel não concordava com as afirmações de Church, pois para ele faltava clareza e justificativa. Como resposta ao questionamento de Church sobre a visão dele de efetivamente calculável, Gödel desenvolveu a definição baseando-se nas propostas de Herbrand e ajustando-as, assim, criando a recursividade de Gödel-Herbrand, simplesmente conhecida como funções recursivas (YANDELL, 2002).

Em 1935, Kleene e Church, em trabalhos independentes, provaram que o conceito de λ-definível é equivalente à recursividade de Gödel-Herbrand (KLEENE, 1936). No ano seguinte, Church publicaria a sua tese, apresentando a resposta negativa ao *Entscheidungsproblem* (CHURCH, 1936).

De 1935 a 1936 e alheio aos trabalhos de Church, Gödel e Kleene, Alan Turing estava trabalhando em outra abordagem para a resolução do *Entscheidungsproblem*, ele apresentou um modelo teórico de máquinas de computar, que mais tarde seriam chamadas de máquinas de Turing. Paralelo às funções que podem ser efetivamente calculáveis, as fórmulas ou funções que poderiam ser computadas na máquina hipotética seriam fórmulas ou funções efetivamente computáveis. Quando soube que o problema já havia sido resolvido um pouco antes da publicação de seu artigo sobre as máquinas de computar, Turing adicionou um apêncide, apresentando que ambos os conceitos de λ-definível e efetivamente computável são equivalentes (TURING, 1937b).

Em 1943, Kleene propôs o termo "tese de Church" em seu artigo *Recursive Predicates and quantifiers*:

> Este fato heurístico [funções recursivas são efetivamente calculáveis] [...] levou Church a formular a seguinte tese. A mesma tese é implícita na descrição de Turing de máquinas de computação:
>
> *TESE I. Toda função efetivamente calculável [...] é recursiva geral.*
>
> Assim como uma definição precisa do termo efetivamente calculável (efetivamente decidível) é desejada, podemos usar essa tese [...] como definição dela [...] (KLEENE, 1943, tradução nossa) (DAVIS, 1965).[19]

[19]Tradução livre do trecho: "This heuristic fact [...] led Church to state the following thesis.

Uma tese, que pode ser uma conjectura ou hipótese, não tem a possibilidade de provar a sua veracidade formalmente, porém, ela ainda pode ser aceita e usada em outros meios e domínios. Logo, se for possível mostrar que é possível calcular alguma função sem o uso de recursão geral, então a tese de Church é provada falsa, no entanto, não há como provar que a tese é verdadeira pela impossibilidade de contar todas as funções calculáveis (YANDELL, 2002).
Em 1952 na publicação do livro *Introduction to Metamathematics*, no capítulo relacionado às funções computáveis, Kleene apresenta as duas teses, de Church e de Turing, e mostra a equivalência de ambas, assim, vindo a nomear a tese de Church-Turing.

2.2 CÁLCULO LAMBDA

O Cálculo lambda é um sistema formal criado por Alonzo Church com o propósito de formular as bases da lógica e partes da matemática, como também promover uma teoria geral das funções. Apesar de inconsistências encontradas na tentativa de formular as bases da lógica[20], o sistema formal estava cumprindo o requisito de formular uma teoria geral das funções. Com isso, definiu o conceito de "efetivamente calculável", assim, fornecendo a resposta negativa ao *Entscheidungsproblem*, problema proposto por Hilbert que perguntava sobre a existência de alguma forma de decidir a provabilidade de qualquer fórmula de lógica de primeira ordem.
Tudo em cálculo lambda são funções, portanto, uma função "é uma regra que associa cada elemento x de um conjunto D a exatamente um elemento, chamado de $f(x)$, em um conjunto E" (STEWART, 2008). O Cálculo Lambda é um modelo teórico de computação sobre funções como regras, ou seja, preza pela definição de regras *genéricas* que realizam a computação através de parâmetros como entrada, portanto, é uma linguagem descritiva ou funcional. Uma forma de classificar linguagens é chamada de paradigmas da programação, dois tipos mais comuns são as funcionais e as imperativas, sendo que as imperativas descrevem uma sequência de passos a serem seguidos, em outras palavras, uma série de comandos (O'DONNEL, 1977).
Sendo uma linguagem formal, o cálculo lambda possui uma gramática, assim,

---

The same thesis is implicit in Turing's description of computing machines.

THESIS I. Every effectively calculable function [...] is general recursive

Since a precise mathematical definition of the term effectively calculable (effectively decidable) has been wanting, we can take this thesis [...] as a definition of it [...]."

[20]Tentativas e progressos foram feitos para formular um sistema formal que cumpra o requisito de forma consistente, porém, o problema ainda está em aberto. Ver referências em (BARENDREGT, 1984, capítulo 1.1) para mais detalhes.

possui um conjunto de produção de regras que apresenta quais as sequências de símbolos válidos de acordo com a sintaxe da linguagem. A seguir é ilustrada a gramática do cálculo lambda, adaptada de (PIERCE, 2002).

$$
\begin{aligned}
&<expressão> ::= \\
&\qquad <variável> \\
&\qquad | \ \lambda \ <variável>.<expressão> \\
&\qquad | \ <expressão> \ <expressão>
\end{aligned}
$$

Uma expressão em cálculo lambda pode ser composta de uma de três possibilidades: uma variável, por exemplo *x*, *y*, etc.; ou uma abstação, que possui a forma de um símbolo λ acompanhado da variável ligada atribuída a uma expressão, que é chamada de corpo da função. Exemplos de abstrações temos "$\lambda x.x$", "$\lambda z.(3z+2)$" e "$\lambda x.(\lambda y.(y^2+x))$"; ou, por fim, uma aplicação, duas outras expressões que podem ser um dos três itens já mencionados.
Um exemplo[21] para ilustrar o paradigma funcional: supondo que queremos realizar a soma dos fatoriais de 8, de 3 e de 4. Uma forma de resolver é realizar as operações diretamente, realizando cálculos repetidamente:

$$(8\times7\times6\times5\times4\times3\times2\times1)+(3\times2\times1)+(4\times3\times2\times1).$$

Outra forma é definir uma função `fatorial(n)`, onde para cada *n* não negativo:

```
fatorial(n) = se n=0 então 1 senão n*fatorial(n-1),
```

e escrever `fatorial(8) + fatorial(3) + fatorial(4)`, então a cada chamada da função o parâmetro *n* produz o fatorial resultante.
Conforme comparado em (PIERCE, 2002), se usarmos "λn...." como "a função que, para cada *n*, produz...", teremos a função `fatorial(n)` como:

```
fatorial(n) = λn. se n=0 então 1 senão n*fatorial(n-1)
```

Logo, a função `fatorial(0)` significa

> "a função (λn. se n=0 então 1 senão n*fatorial(n-1)) cujo argumento é zero",

que significa

> "o resultado da função (λn. se n=0 então 1 senão n*fatorial(n-1)) se *n* for igual a zero", que significa "se 0=0 então 1 senão n*fatorial(n-1)",

[21] Exemplo de (PIERCE, 2002).

que significa “um”.
De um parâmetro numérico aplicado a uma função até o resultado númerico, ou forma normal, se passam por uma série de transformações, ou *reduções*, entre os dois termos, portanto, o cálculo lambda é um sistema de redução e, inclusive, segue a propriedade de Church-Rosser, que define que a forma normal é obtida independente da ordem de cálculo dos subtermos (BARENDREGT, 1994).
O Cálculo Lambda foi a inspiração para as linguagens de programação. As linguagens de paradigma funcional tiveram influência direta do modelo formal, tais como Haskell, Lisp e F#, e indiretamente influenciou as linguagens imperativas, como apontado por Peter Landin a relação entre Cálculo Lambda e expressões em ALGOL 60 (LANDIN, 1965a; LANDIN, 1965b). O ALGOL 60 inspirou diversas linguagens imperativas subsequentes, tais como Pascal e C, sendo esta influenciando a criação de linguagens como C++, Java e Python.
O Haskell é uma linguagem puramente funcional, assim, é notável a influência de cálculo lambda. Por exemplo, em Haskel o símbolo λ é simbolizado como \ e o ponto que separa a variável ligada do corpo da função é simbolizada como ->. Funções lambda em programação são funções anônimas com o propósito de passar o parâmetro para uma função de alta ordem ou realizar tarefas para a função de alta ordem retornar - funções de alta ordem recebem ou retornam funções. A seguir, dois exemplos simples: uma função lambda $(\lambda x.2x)$ e uma função de alta ordem, `map`, que recebe tal função lambda e uma lista de números naturais de 1 a 10 e aplica a função lambda a cada elemento da lista; o segundo exemplo possui a função lambda $(\lambda xy.x+y)$ e a função de alta ordem, `zipWith`, que aplica a função lambda e cada elemento das listas correspondentes, a primeira lista corresponde aos valores de `x` e segunda corresponde aos valores de `y`.

```
Prelude> map (\x -> x*2) [1..10]
[2,4,6,8,10,12,14,16,18,20]

Prelude> zipWith (\x y -> x+y) [5,9,7] [10,9,1]
[15,18,8]
```

2.2.1 Operações básicas

Iremos apresentar um axioma e as duas operações básicas em cálculo lambda: aplicação e abstração. A primeira operação básica é a *aplicação*, por exem-

plo[22], supondo que temos a expressão x^2+x+3, o que resultaria se $x=5$? Neste caso, estamos *aplicando* o parâmetro $x=5$ à expressão. Na notação de cálculo lambda, temos a seguinte função:

$$\lambda x[x^2+x+3](A). \tag{2.1}$$

A função 2.1 mostra a variável x sendo *abstraída*, ou seja, estamos colocando a variável x em perspectiva a fim de avaliar os possíveis resultados ao definirmos algum valor a A que será aplicado em x. Genericamente, podemos definir a função 2.1 como "$F(A)$" ou "$F.A$" ou "FA"[23]. Assim, a abstração é a segunda operação básica. O símbolo de λ serve como um indicador da variável ligada, no caso, a variável x, em notação de função matemática, equivalente a 2.1, temos $f(x)=x^2+x+3$. Se aplicarmos o número 5 à função que abstrai x de "x^2+x+3" temos:

$$\begin{aligned}\lambda x[x^2+x+3](5) &= 5^2+5+3\\ &= 25+5+3\\ &= 30+3\\ &= 33\end{aligned}$$

Do exemplo mostrado, segue o seguinte axioma do cálculo lambda:

$$(\lambda x.M[x])N = M[x := N]. \tag{2.2}$$

Em palavras, "$(\lambda x.M[x])N$" apresenta uma função $M[x]$ qualquer que contém a variável ligada x (indicada pelo λ) e um N qualquer, seja um valor numérico ou alguma função, que será aplicado à função $M[x]$; o lado direito da igualdade, "$M[x := N]$", representa a aplicação de N em $M[x]$, ou seja, todas as ocorrências de x na função $M[x]$ serão substítuidos por N. O axioma citado se chama redução β.

2.2.2 Definições

λ-termos: Apresentada uma expressão formada por variáveis e/ou constantes, os λ-termos são definidos da seguinte forma:

(i) Todas variáveis e constantes são chamadas de λ-termos (definidos como átomos);

[22] Exemplo baseado de (IRVINE; DEUTSCH, 2016)

[23] Como comparado em (BARENDREGT, 1994), podemos dizer que F é um algoritmo aplicado ao dado A, sendo este a entrada.

(ii) Sendo A e B quaisquer λ-termos, então (AB) é um λ-termo (definido como aplicação);

(iii) Sendo A um λ-termo e x uma variável, então $(\lambda x.A)$ é um λ-termo (definido como abstração).

Identidade sintática: Indicado pelo símbolo "$\equiv$", a identidade sintática de, por exemplo A e B, indicada por $A \equiv B$ explicita que A é exatamente o mesmo termo que B.

Tamanho de um termo: O tamanho de um termo M qualquer (denotado como `lgh(M)`) é definido pelo número de átomos que ele possui, definindo-os:

- `lgh(a) = 1` para quaisquer átomos a;
- `lgh(MN) = lgh(M) + lgh(N)`;
- `lgh(λ x.M)` $= 1+$ `lgh(M)`

Ocorrências: A relação P ocorre em Q (ou P é subtermo que Q, ou Q contém P) é definida pela indução em Q, assim:

- P ocorre em P;
- Se P ocorre em M ou N, então P ocorre em (MN);
- Se P ocorre em M ou $P \equiv x$, então P ocorre em $(\lambda x.M)$.

Exemplo: Há duas ocorrências de (xy) e três ocorrências de x em $((xy)(\lambda x.(xy)))$. Não há ocorrência de $x(yz)$ em $ux(yz)$, pois isto é na verdade $((ux)(yz))$.

Escopo: Para uma ocorrência particular de $\lambda x.M$, a ocorrência de M é chamada de escopo da ocorrência de λ à esquerda. Por exemplo, seja expressão:

$$(\lambda y.yx(\lambda x.y(\lambda y.z)x))vw.$$

O escopo do λ mais à esquerda é $yx(\lambda x.y(\lambda y.z)x)$, o escopo do segundo λ partindo da esquerda é $y(\lambda y.z)x$ e, por fim, o escopo do λ mais à direita é z.

Variáveis livres e ligadas: A ocorrência de uma variável x no termo P é dita ligada se, e somente se, x for parte de P da forma $\lambda x.N$, caso contrário, x é variável livre. Se x tem pelo menos uma ocorrência livre em P, então x é chamado de variável livre de P. Se P é um termo sem variáveis livres, ele

é chamado de termo fechado ou **combinador**. A ideia de combinadores é serem um sistema similar ao próprio Cálculo Lambda, mas sem o uso de variáveis ligadas (HINDLEY; SELDIN, 2008). Um exemplo[24] de combinador, queremos gerantir a comutatividade da soma, assim, definindo o combinador de adição A e definindo que para todo x e y temos:

$$A(x,y) = x + y.$$

Especificando agora outro combinador C para garantir a comutatividade e que para todo x, y e f:

$$(C(f))(x,y) = f(y,x).$$

Assim, a regra da comutatividade fica definida como $A = C(A)$. Aplicando um exemplo onde $x = 1$ e $y = 2$:

$$\begin{aligned} A(1,2) &= 1+2 = 3 \\ (C(A))(1,2) &= A(2,1) = 2+1 = 3. \end{aligned}$$

Um combinador particular existente no cálculo lambda é o combinador Y, que é um combinador de ponto fixo. Um ponto fixo de uma função é quando um elemento do domínio é mapeado para si mesmo no contradomínio, por exemplo, a função $f(x) = x^2 - 3x + 4$ contém o ponto fixo 2, pois $f(2) = 2$. O combinador Y foi descoberto por Haskell Curry e é definido da seguinte forma:

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

Assim, o combinador recebe uma expressão que é aplicada em f. Agora, se atribuirmos uma variável g a Y:

$$Yg = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))g.$$

Seguindo a redução β, assim, substituindo as ocorrências de f por g, temos:

$$(\lambda x.g(xx))(\lambda x.g(xx)).$$

Aplicando novamente a redução β, ou seja, substituindo os x da função à esquerda por "$\lambda x.g(xx)$" teremos:

$$g((\lambda x.g(xx))(\lambda x.g(xx))).$$

[24] Exemplo de (HINDLEY; SELDIN, 2008).

O conteúdo dentro dos parêntes é igual ao resultado após a aplicação da variável g em Y, logo:

$$g(Yg).$$

Tal operação de "retorno" realizada, de "$(\lambda x.g(xx))(\lambda x.g(xx))$" para "$Yg$", é possível graças à terceira regra de conversão do cálculo lambda (ver mais adiante). Como é notável, de uma função Yg resultou-se em $g(Yg)$, a aparição de uma nova variável g, resultando num paradoxo. Se continuarmos aplicando g nos resultados, teremos:

$$Yg = g(Yg) = g(g(Yg)) = g(g(g(Yg))) = g(...g(Yg)...).$$

Curry chamou o combinador Y de *combinador paradoxal* (BARENDREGT, 1984). O resultado do combinador Y é similar ao de comandos `for` e `while` em linguagens imperativas e, notadamente, o combinador implementa uma recursão simples.

Conversão: Uma fórmula pode ser convertida a outra, ela deve seguir uma das três propriedades a seguir:

(I) Você pode mudar uma variável ligada (por exemplo, x para y) se a nova variável não interferir com nenhuma outra coisa na fórmula.

(II) Na fórmula $\{\lambda x.M\}(N)$, se N não contém nada nomeado como x, você pode substituir N por todas as ocorrências de x em M, onde a fórmula se torna apenas M com N substituído pelo x original.

(III) O reverso do II é permitido (PETZOLD, 2008).

Uma fórmula pode ser convertida para outra similar, seguindo a regra I, II ou III, simbolizado como "*conv*". Por exemplo:

(I) $\lambda x[x^3 - x + 2](A)$ *conv* $\lambda y[y^3 - y + 2](A)$.

(II) $\lambda x[x^2 + 2x + 17](A)$ *conv* $A^2 + 2A + 17$.

(III) $A^2 + 2A + 17$ *conv* $\lambda x[x^2 + 2x + 17](A)$.

2.2.3 Outras aplicações

Uma aplicação que é possível trabalhar com cálculo lambda é a lógica booleana. Assim, definimos dois valores verdade T e F como:

$$T \equiv \lambda xy.x,$$
$$F \equiv \lambda xy.y$$

A função T ou *True* possui dois parâmetros de entrada, mas retorna apenas o primeiro, tal qual a função F ou *False*, mas retorna o segundo parâmetro.
Com os valores verdades definidos, é possível criar uma função condicional, que chamaremos de IF:

$$IF \equiv \lambda b.(\lambda x.(\lambda y.(bxy))).$$

Da função IF, se o condicional for verdadeiro, então retorna x, caso contrário (cláusula *else*) retornará y. A seguir um exemplo ilustrando o uso do seguinte condicional: "se T então p, senão q" - a resposta esperada é p.

$$\begin{aligned}
\boldsymbol{IF}\,T\,pq &= (\lambda b.(\lambda x.(\lambda y.(bxy))))T\,pq \\
&= (\lambda b.(\lambda x.(\lambda y.(bxy))))(\lambda xy.x)pq \\
&= (\lambda x.(\lambda y.(\lambda xy.(x)xy)))pq \\
&= (\lambda y.(\lambda xy.(x)py))q \\
&= (\lambda xy.(x)pq) \\
&= (\lambda xy.x)(pq) \\
&= p
\end{aligned}$$

Outras duas operações básicas que podemos definir são o AND e o OR:

$$AND \equiv \lambda x.(\lambda y.xyx),$$
$$OR \equiv \lambda x.(\lambda y.xxy).$$

A seguir temos um exemplo de uma operação AND entre os dois valores-

verdade T e F:

$$\begin{aligned}
&\boldsymbol{AND}TF \\
&= (\lambda x.(\lambda y.xyx))TF \\
&= (\lambda x.(\lambda y.xyx))(\lambda xy.x)F \\
&= (\lambda x.(\lambda y.xyx))(\lambda xy.x)(\lambda xy.y) \\
&= (\lambda y.(\lambda xy.x)y(\lambda xy.x))(\lambda xy.y) \\
&= (\lambda xy.x)((\lambda xy.y)(\lambda xy.x)) \\
&= (\lambda xy.y) \\
&= F.
\end{aligned}$$

Um exemplo de uma operação OR entre os dois valores-verdade T e F:

$$\begin{aligned}
&\boldsymbol{OR}FT \\
&= (\lambda x.(\lambda y.xxy))FT \\
&= (\lambda x.(\lambda y.xxy))(\lambda xy.y)T \\
&= (\lambda x.(\lambda y.xxy))(\lambda xy.y)(\lambda xy.x) \\
&= (\lambda y.(\lambda xy.y)(\lambda xy.y)y)(\lambda xy.x) \\
&= ((\lambda xy.y))(\lambda xy.y)(\lambda xy.x) \\
&= (\lambda xy.x) \\
&= T.
\end{aligned}$$

2.3 MÁQUINAS DE TURING

Uma máquina de Turing é um modelo téorico de uma máquina de computar, a estrutura de uma máquina, ilustrada na figura 1, contém uma fita infinita composta por vários campos (ou "quadrados") - a fita da máquina serve como a sua memória, assim, se algo necessita ser "salvo", é preciso escrever nela. A máquina possui uma cabeça de fita ou cabeçote que é capaz de percorrer a fita nas direções à esquerda e à direita.
O processo de computação da máquina é definido por uma sequência de *estados* finitos da máquina, representando o controle finito que possui um identificador apontando para o estado atual. O conceito de *estado*[25] em máquinas de Turing é ambíguo, e como esclarecido em (HODGES, 1983, p. 137), pode significar duas coisas:

[25] "*Configuração*" também usado como sinônimo para "estado"

1. A *configuração interna* da máquina, ou seja, um estado é composto pelo o conteúdo da fita, onde o cabeçote está posicionado, em qual valor está sendo apontado pelo cabeçote, qual o valor a ser escrito no campo apontado pelo cabeçote, etc.;

2. Uma *instrução*, portanto, um estado apenas descreve qual tarefa a máquina deve fazer, por exemplo, escrever algo na fita ou mover o cabeçote para uma determinada direção.

Para fins de clareza, iremos definir *estados* conforme descrito no primeiro item.

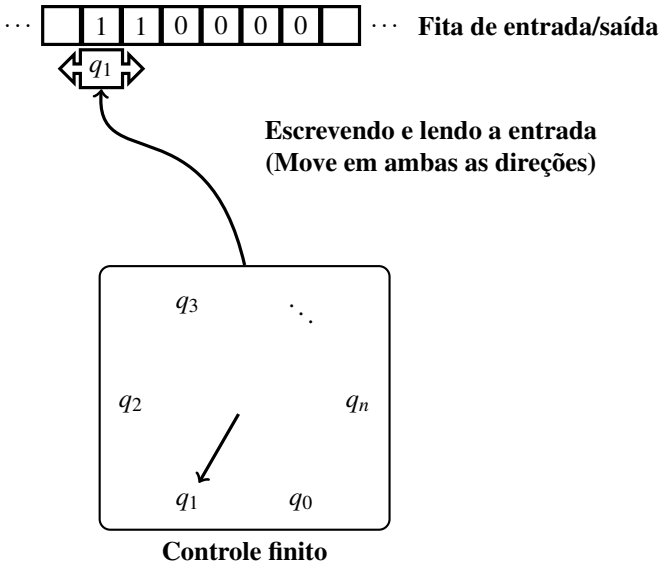


Figura 1: Representação de uma máquina de Turing.

2.3.1 Computação de problemas

As máquinas realizam o processo de computar, ou seja, ela realiza uma sequência de passos, seja eles estados ou instruções, seguindo um modelo bem-definido, um exemplo de tal modelo é um algoritmo, que descreve cada operação ou cálculo que uma máquina deve fazer. O resultado da computação da máquina de Turing depende bastante do problema (a entrada) que foi designado a ela, portanto, a máquina pode retornar como resultado (a saída)

uma de duas possibilidades: a máquina é uma decisora, dado um problema de decisão, a máquina pode decidir sobre ele, com uma resposta do tipo “sim” ou “não”; a máquina consiste numa função, assim, dependendo da entrada a máquina gera um resultado associado.
Detalhadamente à primeira possibilidade, de uma máquina de decidir, se a máquina retorna uma resposta do tipo “sim” ou “não”, então ela *decidiu* sobre o problema; caso a máquina não chegue a nenhuma resposta, apesar de *reconhecer* o problema, então ela ficará executando infinitamente, não informando nenhum resultado específico. Assim, destas duas ocasiões, os problemas que resultam numa resposta exata, são os problemas *decidíveis*[26], por outro lado, os problemas em que a máquina processa infinitamente, não chegando a um resultado, são problemas *reconhecíveis*.[27] Note que todo problema decidível é também reconhecível, mas o contrário não necessariamente é verdadeiro.
Os problemas de decisão foram um dos principais fatores para a criação das máquinas de Turing, pois o problema de decisão proposto por Hilbert, o *Entscheidungsproblem*, buscava determinar se uma dada fórmula é resolvível ou não e a motivação da criação das máquinas de Turing veio disto. Turing viera a mostrar que não é possível determinar e, ainda, não teria como garantir que a máquina que resolvesse a fórmula pararia em algum momento (se entraria num *loop* infinito), assim, temos o *problema da parada*.

2.3.2 Definição formal

De acordo com a definição de (HOPCROFT; MOTWANI; ULLMAN, 2001) e (SIPSER, 2013)[28], uma máquina de Turing M é uma séctupla tal que

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

onde

1. Q é o conjunto finito de estados definidos no controle;

2. Σ é o conjunto finito de símbolos de entrada, em outras palavras, é o alfabeto de entrada não contendo o símbolo de vazio B;

[26] Dependendo da bibliografia, há o uso dos sinônimos de “problemas decidíveis": *Turing-decidíveis* ou *linguagens recursivas*.

[27] Dependendo da bibliografia, há o uso dos sinônimos de “problemas reconhecíveis": *Turing-reconhecíveis* ou *linguagens recursivamente enumeráveis*.

[28] Em (SIPSER, 2013) é definido que uma máquina de Turing M é tal que $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, onde q_{accept} representa o estado final ou de aceitação da máquina e q_{reject} representa o estado de rejeição da máquina, ou seja, o estado em que a máquina retorna um resultado inválido ao problema.

3. Γ é o alfabeto da fita, onde contém o símbolo vazio B e Σ sempre é subconjunto de Γ;
4. δ é a função de transição, que define como será a passagem de estados da máquina. A função $\delta(q,X)$, onde q é um estado em Q e X é um símbolo do alfabeto da fita em Γ, resulta em (r,Y,R), onde:
   - r é o próximo estado e r é elemento de Q;
   - Y é um símbolo, em Γ, a ser escrito no campo da fita em que está selecionado, sobrescrevendo o que estiver escrito antes[29];
   - R é a direção onde a cabeça de fita irá, R faz o cabeçote se mover à direita (*right*) e L faz o cabeçote se mover à esquerda (*left*).
5. q_0 é o estado inicial da máquina e elemento do conjunto Q;
6. B é o símbolo vazio. O símbolo pertence a Γ, mas não pertence a Σ. O símbolo de vazio aparece inicialmente em todos os campos da fita;
7. F é o conjunto de estados finais ou de aceitação, ou seja, estados onde a máquina retorna um resultado válido ao problema proposto. F é um subconjunto de Q.

2.3.3 Exemplos de máquinas de Turing

Esta seção irá apresentar dois exemplos com as respectivas definições das máquinas de Turing usadas para lidar com os problemas propostos.

2.3.3.1 Exemplo 1

> Construir uma máquina que seja capaz de reconhecer a sequência $(01)^n$, onde $n \geq 0$.

A tarefa da máquina é bem simples, ela deve reconhecer uma subcadeia de números formada apenas por `01`, logo, ela pode reconhecer valores como `01` e `0101010101` - quaisquer valores que não sejam do formato são inválidos. Assim, uma cadeia w está escrita na fita, e a máquina, cujo estado inicial tem o cabeçote apontando para o primeiro símbolo mais à esquerda de w, percorrerá por toda a cadeia w e decidirá se tal cadeia é válida ou não.

[29] Emil Post (1897-1954) escreveu no apêndice de sua publicação "*Recursive Unsolvability of a Problem of Thue*" algumas convenções que são mais restritas sobre o comportamento das máquinas. Dentre elas, uma máquina nunca apaga um campo numérico ou sobrescreve uma figura existente num campo numérico com outra figura (PETZOLD, 2008, p.86).

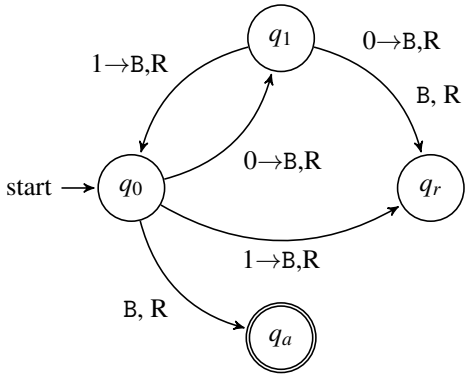


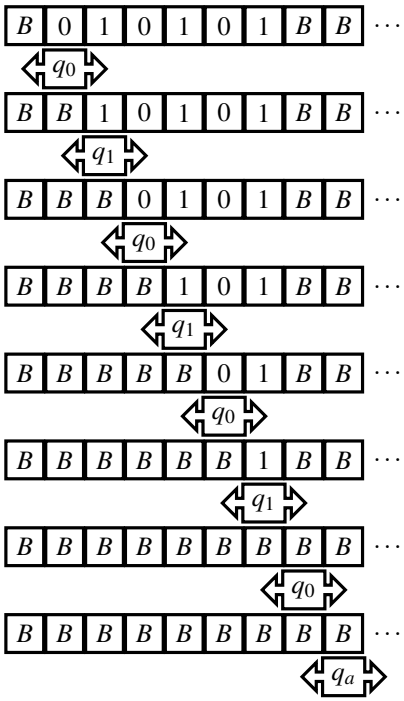
Figura 2: Função de transição da máquina.

A computação para o reconhecimento da cadeia w está apresentada no diagrama de estados apresentado na figura 2. O começo está no estado q_0 que pode ir tanto para o estado de aceitação (q_a) ou para o estado q_2: Se for para q_a, isto indica que a cadeia w é vazia, o que é um valor válido, portanto, o reconhecimento para e a máquina decide sobre w; se for para q_2 indica que há um símbolo não vazio e o processo de reconhecimento deve continuar. O identificador "0→B, R" indica que a máquina leu e reconheceu 0 e irá substituir pelo símbolo vazio B, em seguida, irá mover o cabeçote para a direita (*right*), enquanto "B, R" indica que a máquina leu e reconheceu B e move o cabeçote para a direita.
Estando em q_2, o elemento da cadeia w será avaliado, se ele conter um símbolo 1, então voltará para q_0, porém, se conter um símbolo 0 ou o símbolo vazio B, então irá decidir que é uma cadeia inválida, irá para o estado de rejeição e a máquina para a computação. Voltando para q_0, se o próximo elemento de w for B, isso indica que $w = 01$ e é um valor válido, caso contrário, a máquina continuará computando, até terminar a cadeia e decidir se é uma cadeia válida ou não.
Formalmente, a máquina pode ser descrita como uma máquina de Turing M_1 que é uma séctupla $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, onde:

- $Q = \{q_0, q_1, q_a, q_r\}$;
- $\Sigma = \{0, 1\}$;
- $\Gamma = \{0, 1, B\}$;
- A função de transição δ é descrita na figura 2;
- O estado inicial é q_0;

- O estado de aceitação é definido como $F = \{q_a\}$.

A seguir, uma ilustração de passo a passo, de acordo com a figura 2, para reconhecer a sequência `010101`.



2.3.3.2 Exemplo 2

> Descrever uma máquina que seja capaz de computar a sequência 001011011101111...[30]

Para computar a sequência especificada, como é notável, precisamos saber o número de vezes que o símbolo `1` foi escrito da última vez para acrescentar mais um na sequência seguinte. A tabela 2 a seguir descreve como a computação da máquina M_2 deve ser realizada. Das operações: `P` escreve o valor na fita; `E` apaga o que está escrito no campo apontado na fita; `L` move o cabeçote para a esquerda; `R` move o cabeçote para a direita.

[30] Exemplo de (TURING, 1937b)

estado	símbolo	operações	próximo estado
q_0		Pe, R, Pe, R, P0, R, R, P0, L, L	q_1
q_1	1	R, Px, L, L, L	q_1
	0		q_2
q_2	`0` ou `1`	R, R	q_2
	Nenhum	P1, L	q_3
q_3	`x`	E, R	q_2
	`e`	R	q_4
	Nenhum	L, L	q_3
q_4	`0` ou `1`	R, R	q_4
	Nenhum	P0, L, L	q_1

Tabela 2: O funcionamento da máquina.

O sequênciamento deve ser feito da seguinte forma: após a impressão da sequência inicial `0010` (tendo um espaço entre cada dígito para a contagem da quantidade de número 1), é impresso `1` como próximo elemento da sequência, após isso, o cabeçote volta para começar a contar quantos `1` já foram escritos, escrevendo `1` e apagando um `x` correspondente, a fim de garantir que a subsequência seguinte contenha um elemento `1` a mais da subsequência anterior, assim, imprimindo `0010110`.
A definição formal da máquina M_2 é uma séctupla $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, onde:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$;
- $\Sigma = \{0, 1\}$;
- $\Gamma = \{0, 1, B, e, x\}$;
- A função de transição δ é descrita na tabela 2;
- O estado inicial é q_0;
- Não há estados de aceitação, assim, $F = \{\}$.

Como é possível notar, a máquina consegue computar tal sequência, mas ela nunca irá parar seu processamento. O problema proposto para esse exemplo é *reconhecível*, pois uma máquina de Turing é capaz de executá-lo, mas ele não é *decidível*, ou seja, a máquina não consegue parar e retornar uma resposta (*sim* ou *não*, *aceita* ou *rejeita*).

2.3.4 *Skeleton tables*

Quando apresentado pela primeira vez o modelo teórico de uma máquina de computar em *Computable* (TURING, 1937b), a forma usada para programar na máquina foram as *skeleton tables*, que são tabelas similares à tabela 2. As *skeleton tables* são tabelas que listam as configurações usadas pela máquina para computar o que for definido nelas, no exemplo 2.3.3.2 a tabela descreve um *programa* para escrever a sequência "001011011101111...". A tabela 3 ilustra um exemplo de uma função de busca de um símbolo *x* pela fita[31].

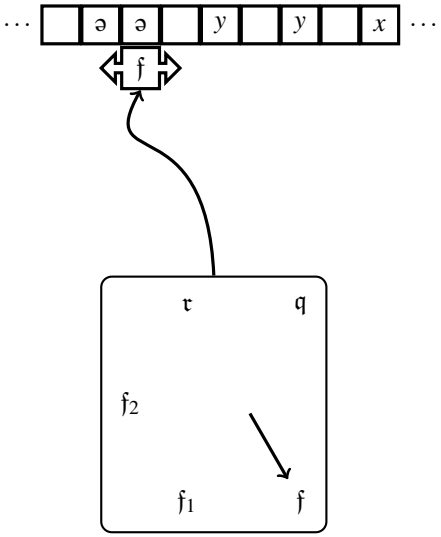


Figura 3: Representação de uma máquina de Turing que computa a tabela 3.

Uma *skeleton table* é composta por quatro colunas: *m-config* ou configuração de máquina, corresponde à configuração ou ao estado atual da máquina; *símbolo*, corresponde ao símbolo sendo apontado pelo cabeçote; *operações*, apresenta qual operação a máquina deve fazer, seja mover o cabeçote para a esquerda (L) ou para a direita (R); *m-config final* ou configuração de máquina final, representa o próximo estado ou configuração da máquina.
Começando em ɟ, a procura é feita movendo o cabeçote pela esquerda, à procura pelo símbolo *xevá* (ə) que representa o começo da sequência, ao encontrar, é realizada a procura pelo símbolo *x*, movendo o cabeçote para a direita. Dois espaços vazios consecutivos representam o fim da sequência, assim, não

[31] Tabela adaptada de (PETZOLD, 2008, p. 116).

m-config	*símbolo*	*operações*	*m-config final*
$\mathfrak{f}$	ə	L	$\mathfrak{f}_1$
	não ə	L	$\mathfrak{f}$
$\mathfrak{f}_1$	*x*		$\mathfrak{q}$
	não *x*	R	$\mathfrak{f}_1$
	Nada	R	$\mathfrak{f}_2$
$\mathfrak{f}_2$	*x*		$\mathfrak{q}$
	não *x*	R	$\mathfrak{f}_1$
	Nada	R	$\mathfrak{r}$

Tabela 3: *Skeleton table* representando uma função de busca.

há *x* na sequência e a configuração final passa a ser $\mathfrak{r}$, por outro lado, quando um *x* é encontrado, a configuração final passa a ser $\mathfrak{q}$. Uma observação que "não *x*" representa qualquer símbolo não branco que não seja *x*, enquanto "nada" significa que o espaço está em branco. A figura 3 ilustra uma máquina de Turing que computa a função definida na tabela 3: a configuração atual é $\mathfrak{f}$, assim, o cabeçote será movido para a esquerda e a configuração interna da máquina passa a apontar para $\mathfrak{f}_1$ e a computação segue até a máquina chegar num estado de aceitação quando achar o *x* ($\mathfrak{q}$). A seção 3.3 apresenta as *skeleton tables* usadas por Turing para a criação de uma máquina para computar funções equivalentes a funções λ.

3 EQUIVALÊNCIA ENTRE CL E MT

Este capítulo busca apresentar e detalhar cada ponto da publicação "*Computablity and λ-definability*" (TURING, 1937a) de Alan Turing, onde, apresenta como os modelos teóricos máquina de Turing e Cálculo λ são equivalentes. Para fins de compreensão e sem a possibilidade de dubiedades, os exertos do artigo aqui apresentados estarão no seu idioma original, em inglês.

3.1 INTRODUÇÃO

> Several definitions have been given to express an exact meaning corresponding to the intuitive idea of 'effective calculability' as applied for instance to functions of positive integers. The purpose of the present paper is to show that the computable [...] functions introduced by the author are indentical with the λ-definable [...] functions of Church and the general recursive functions due to Herbrand and Gödel and developed by Kleene. [...] (TURING, 1937a)[1]

Computablity and λ-definability foi publicado em dezembro de 1937, Turing se propôs a detalhar a prova de equivalência da Máquina de Turing com o Cálculo Lambda. Antes, ele havia adicionado um apêncice à publicação *On Computable Numbers, With An Application To The Entscheidungsproblem* (TURING, 1937b), onde descreveu e definiu o que seriam mais tarde chamadas de Máquinas de Turing, apresentando a ideia do porquê ambos os conceitos seriam equivalentes.

Para mostrar que ambos os modelos são equivalentes, é necessário mostrar que dado um conjunto *A* formado de todas as possíveis entradas a serem computadas por Máquinas de Turing e outro conjunto *B* formado de todas as possíveis entradas válidas a serem calculadas por funções em Cálculo λ, ambos os conjuntos devem ser iguais. Os elementos do conjunto *A* definem os problemas que são "efetivamente computáveis", conforme detalhado por Turing em (TURING, 1937b), enquanto os elementos do conjunto *B* definem o problemas ou funções que são λ-definíveis, proposto por Church em (CHURCH, 1936). Stephen Kleene provou que λ-definível é equivalente às funções re-

[1] Tradução do autor: Diversas definições foram dadas para expressar um exato significado da intuitiva ideia de 'efetiva calculabilidade' como aplicado por exemplo a funções de inteiros positivos. O propósito deste trabalho é apresentar que funções computáveis apresentadas pelo autor são identicas às funções λ-definíveis de Church e às funções recursivas gerais de Herbrand e Gödel e desenvolvida por Kleene.

cursivas que foram apresentadas por Kurt Gödel e Jacques Herbrand.

> [...] There is a modified form of λ-definability, known as λ-K-definability, and it turns out to be natural to put the proof that every λ-definable function is computable in the form of a proof that every λ-K-definable function is computable; that every λ-definable function is λ-K-definable is trivial. If these results are taken in conjunction with an already available [...] proof that every general recursive function is λ-definable we shall have the required equivalence of computability with λ-definability and incidentally a new proof of the equivalence of λ-definability and λ-K-definability. [...] (TURING, 1937a) [2]

A ser detalhado mais adiante, a versão modificada de λ-definível, chamada de λ-K-definível, em sua base usa um cálculo λ mais restrito em relação ao usual cálculo λ - ainda assim, ambos as formas de definição são equivalentes entre si e às funções recursivas. Portanto, Turing conclui que, apresentando que "efetivamente computável" é equivalente à λ-K-definível então, indiretamente, também apresenta-se uma nova prova de equivalência entre λ-definível e λ-K-definível.

> [...] The identification of 'effectively calculable' functions with computable function is possibly more convincing than an identification with the λ-definable or general recursive functions. For those who take this view the formal proof of equivalence provides a justification for Church's calculus and allows the 'machines' which generate computable functions to be replaced by the more convenient λ-definitions. (TURING, 1937a) [3]

Turing observa que as notações a serem usadas para descrição da prova formal vão ser as mesmas que ele usou na publicação (TURING, 1937b), como também cita conhecimento necessário às funções computáveis recursivas, que

[2] Tradução do autor: [...] Aqui está uma modificação de λ-definibilidade, conhecida como λ-K-definibildade, e assim se torna mais natural apresentar a prova que cada função λ-definível é computável na forma de uma prova que cada função $\lambda - K$-definível é computável; que cada função λ-definível é uma função $\lambda - K$-definível é trivial. Se esses resultados estão de acordo com a prova já existente de que cada função recursiva geral é λ-definível teremos a equivalência de computabilidade com λ-definibilidade e incidentemente uma nova prova da equivalência de λ-definibilidade e $\lambda - K$-definibilidade. [...]

[3] Tradução do autor: [...] A identificação de funções 'efetivamente calculáveis' com função computável é possivelmente mais convincente do que uma identificação com as funçõs λ-definíveis ou geral recursiva. Para aqueles que pensam nessa visão de prova formal de equivalência proveem uma justificativa para o cálculo de Church e permite que 'máquinas' que geram funções computáveis sejam substituídas pelas λ-definições mais convenientes.

serão detalhadas mais adiante. Por fim, para uma visão geral de como será a prova formal da equivalência entre λ-definível e "efetivamente computável": será proposta uma máquina que é capaz de gerar funções computáveis e tais funções serão traduzidas para funções válidas em Cálculo λ que são λ-definíveis.

3.2 DEFINIÇÃO DE λ-K-DEFINÍVEL

A definição da versão mais restrita de λ-definível foi criada a fim de que seja mais fácil e prático trabalhar em uma máquina. No geral, vão ser três diferenças notáveis entre as duas definições:

1. Uso exclusivo de colchetes `[]`, sem usos aos parênteses `()` ou chaves `{}`;
2. As variáveis são definidas como x^I, x^{II}, x^{III}, ..., sendo que o indicador ^{I} pode ser repetido quantas vezes for necessário, ao invés de letras latinas minúsculas (a, b, c, ...);
3. Mudança na forma da transformabilidade, no caso, a definição de conversão de uma fórmula para a outra não é modificada, apenas muda o visual devido às restrições descritas nos dois itens anteriores.

Dadas as restrições, assim, os símbolos válidos para o novo conceito que será usado pela máquina são λ, `x`, ^{I}, `[` e `]`, em outras palavras, estes símbolos irão compôr o *alfabeto* da linguagem λ-K-definível. Os símbolos definidos serão usados para formar fórmulas, as fómulas válidas serão chamadas de "*fórmulas bem formadas*"[4]. Antes de definir as condições para transformação (ou conversão) de fórmulas, Turing ainda adiciona:

> Also if M and N are properly-formed formulae, then $[M][N]$ (i.e. the sequence consisting of [followed by M then by], [and the sequence N, and finally by]) is a properly-formed formula. If M is a properly-formed formula and V is a variable, then $\lambda V[M]$ is a properly-formed formula. If any sequence is a properly-formed formula it must follow that it is so from what has already been said (TURING, 1937a). [5]

[4] Tradução livre de "*Properly-formed formulae*".

[5] Tradução do autor: Também se M e N são fórmulas bem formadas, então $[M][N]$ (i.e. a sequência contendo [seguido por M então por], [e a sequência N e finalmente por]) é uma fórmula bem formada. Se M é uma fórmula bem formada e V é uma variável, então $\lambda V[M]$ é uma fórmula bem formada. Se qualquer sequência é uma fórmula bem formada esta precisa seguir pela definição já especificada.

Uma nota de rodapé menciona que as letras maiúsculas significarão variáveis ou qualquer fórmula indeterminada. Ilustrando, se M e N forem, respectivamente, $\lambda x^I[x^I]$ e $\lambda x[x]$, então $[\lambda x^I[x^I]][\lambda x[x]]$ é uma fórmula bem formada. Sendo V uma variável, digamos, portanto, x^{II}, então $\lambda x^{II}[\lambda x^I[x^I]]$ também é uma fórmula bem formada.
Em seguida, Turing define que uma fórmula bem formada M vai ser *imediatamente transformável* em N se uma das três condições a seguir for satisfeita:

(i) M é da forma $\lambda V[X]$ e N é da forma $\lambda U[Y]$ onde Y é obtido de X trocando a variável V pela variável U, sendo que não haja ocorrências de U em X;

(ii) M é da forma $[\lambda V[X]][Y]$ onde V é uma variável e N é obtido substituindo V por Y em X. Existem restrições quanto à este item, porque se W for a variável V ou uma variável em Y, então não pode haver λW em X;

(iii) N é imediatamente transformável em M como citado no item (ii).

Das restrições apresentadas, Turing apresenta o conceito sinônimo a *imediatamente transformável*: *imediatamente convertível*. Assim, se a fórmula bem formada A for imediatamente transformável para B, então A é *imediatamente convertível* a B. Disto define-se o conceito de *conversão*: A é *convertível* a B (ou "A *conv* B") partindo de A e realizando uma sequência finita de transformações para fórmulas bem formadas chega-se em B.
Por fim, para ilustrar o conceito de convertibilidade, Turing usa dois exemplos. O primeiro exemplo apresenta o primeiro teorema da aritmética de Peano, o conceito de sucessor. Em 1935, Kleene publicou o desenvolvimento de um sistema lógico formal construído puramente de cálculo lambda, um dos trabalhos realizados na publicação foi das definições de números naturais e do primeiro teorema da aritmética de Peano usando apenas funções lambda (KLEENE, 1935)[6]. Turing adaptou as funções criadas por Kleene de acordo com as restrições exigidas no calculo $\lambda - K$. Para começar, é necessária à definição um número natural inicial, que neste caso será o zero e sua expansão será definida como:

$$\lambda x[\lambda x^I[x^I]].$$

A função sucessor S é definida da seguinte forma:

$$\lambda x^{II}[\lambda x[\lambda x^I[[x][[[x^{II}][x]][x^I]]]]].$$

[6]Na publicação de Kleene, a fórmula $\lambda fx.f(x)$ é abreviada para 1, $\lambda\rho fx.f(\rho(f,x))$ é a função sucessor e $\lambda fx.f(f(x))$ é abreviado para 2.

O resultado da aplicação da função $[S][0]$, portanto, deve resultar na fórmula que abreviada corresponde ao número 1. Assim, substituindo as respectivas fórmulas de $[S][0]$ temos:

$$\mathbf{[}\lambda x^{II}[\lambda x[\lambda x^{I}[[x][[[x^{II}][x]][x^{I}]]]]]\mathbf{]}\mathbf{[}\lambda x[\lambda x^{I}[x^{I}]]\mathbf{]}.$$

A fórmula abreviada como 0 sendo a entrada da fórmula S, a fórmula 0 é aplicada à variável associada ao lambda mais externo, ou seja, todas as ocorrências de x^{II} são substituídas por 0:

$$\lambda x[\lambda x^{I}[[x][[\mathbf{[}\boldsymbol{\lambda x[\lambda x^{I}[x^{I}]]]}\mathbf{[}\boldsymbol{x}\mathbf{]}][x^{I}]]]].$$

Na função lambda mais interna, vemos uma aplicação da variável x à fórmula 0, nesta o lambda mais externo é aplicado, no entanto, não há nenhuma aplicação de x:

$$\lambda x[\lambda x^{I}[[x][\mathbf{[}\boldsymbol{\lambda x^{I}[x^{I}]]}\mathbf{[}\boldsymbol{x^{I}}\mathbf{]}]]].$$

Ainda há mais uma aplicação, a da variável x^{I} na função lambda mais interna:

$$\lambda x[\lambda x^{I}[[x][[[x^{I}]]]]].$$

Sem mais aplicações a serem a realizadas, assim, chega-se ao resultado de $[S][0]$, abreviada como 1. Apresentamos que a fórmula $[S][0]$, após uma série de transformações de uma fórmula bem formada para outra, terminou na fórmula abreviada como 1, ou seja, $[S][0]$ converte para $\lambda x[\lambda x^{I}[[x][[[x^{I}]]]]]$ (simbolicamente, $[S][0]$ *conv* $\lambda x[\lambda x^{I}[[x][[[x^{I}]]]]]$). Sucessivamente, temos que $[S][1]$ converte para $\lambda x[\lambda x^{I}[[x][[x][x^{I}]]]]$ (abreviado como 2), etc..

> A function $f(n)$ of the natural numbers, taking natural numbers as values will be said to be $\lambda - K-$-definable if there is a formula F such that $[F][\mathbf{n}]$ is convertible to the formula representing $f(n)$ if $\mathbf{n}$ is the formula representing n. The formula $[F][n]$ can never be convertible to two formulae representing different natural numbers, for it has been shown [...] that if two properly-formed formulae are in normal form (i.e. have no parts of the form $[\lambda V[M]][N]$) and are convertible into one another, then the conversion can be carried out by the use of (i) only. The formulae representing the natural numbers are in normal form and the formulae representing two different natual numbers are certainly not convertible into one another by use of (i) alone (TURING, 1937a). [7]

[7] Tradução do autor: Uma função $f(n)$ dos números naturais, usando números naturais como

Neste segundo exemplo, Turing apresenta uma função $f(n)$ dos números naturais e esta pode ser dita $\lambda - K$-definível se existir uma fórmula F que faça com que $[F][\mathbf{n}]$ seja convertida para uma fórmula em Cálculo $\lambda - K$ que representa $f(n)$, inclusive, $\mathbf{n}$ é a fórmula em Cálculo $\lambda - K$ representando n (não necessariamente estes dois n são iguais), tal conversão é válida devido à propriedade (i) das conversões. Em seguida, outro ponto destacado é que, dado que as propriedades de conversão do Cálculo $\lambda - K$ serem baseadas nas propriedades de conversão do Cálculo λ de Church, não é possível que a fórmula $[F][n]$ converta em duas fórmulas diferentes representando dois números naturais diferentes, como mostrado por Church e Rosser (CHURCH; ROSSER, 1936), que se duas fórmulas bem formadas estão em suas respectivas formas normais e são convertíveis entre si, então é possível haver uma conversão de uma para outra usando apenas a propriedade (i) da conversão[8]. Por outro lado, se duas fórmulas, uma sendo a dos números naturais e a outra representando dois números naturais diferentes, então é de certeza que ambas as fórmulas não são convertíveis entre si usando apenas a propriedade (i).

3.3 ABREVIAÇÕES

Após as definições de Cálculo $\lambda - K$, agora é necessário criar uma máquina que seja capaz de computar o modelo formal. Toda a formalidade da máquina Turing já detalhou em *Computable* (TURING, 1937b) e é discutida no capítulo 2.3, usando as mesmas nomenclaturas, assim, nesta seção Turing passa apenas a definir as novas funções que serão necessárias para a criação da máquina computar de Cálculo $\lambda - K$. Ao ter as funções, elas poderão ser usadas para definir as conversões, que são as operações básicas de Cálculo $\lambda - K$ e são o tópico da seção 3.4. Antes de tudo isso, porém, iremos apresentar alguns pontos necessários para compreensão da descrição do funcionamento de uma máquina por Turing.

O funcionamento e o visual de uma máquina de Turing já foi explicado na seção 2.3. Em *Computable* (TURING, 1937b), o recurso que Turing usa para

valores, será $\lambda - K$-definível se existe uma fórmula F tal que $[F][\mathbf{n}]$ é convertível à fórmula representando $f(n)$ se $\mathbf{n}$ é a fórmula que representa n. A fórmula $[F][n]$ nunca pode ser convertível a duas fórmulas representando diferentes números naturais, para isto tem que ser mostrado [...] que se duas fórmulas bem formadas estão na forma normal (i.e. não contêm partes da forma $[\lambda V[M]][N]$) e são convertíveis entre si, então a conversão pode ser feita fazendo uso de (i) apenas. A fórmula representando os números naturais é em forma normal e a fórmula representando dois números naturais diferentes não são convertíveis entre si pelo uso de (i) apenas.

[8] Esta conclusão foi obtida a partir do teorema 1 e do corolário 2 da publicação de Church e Rosser; o teorema 1 descreve que "*se A conv B, então tem uma conversão de A para B em que nenhuma expansão precede uma redução*" e o corolário 2 descreve que "*Se A tem uma forma normal, então sua forma normal é única (para aplicações da regra I [da convertibilidade])*".

descrever o funcionamento da máquina é chamado de *skeleton tables* (apresentadas na seção 2.3.4) segundo o próprio, estas por si não são essenciais, são como abreviações[9], porém, são úteis para facilitar o entendimento do funcionamento e são mais fáceis de construir - como apontado por (PETZOLD, 2008), as *skeleton tables* tem um papel importante para facilitar a construção da máquina universal de Turing[10].
As *skeleton tables* são tabelas similares ao segundo exemplo da subseção 2.3.3.2: quatro colunas apresentando as etapas e, horizontalmente, representando o passo-a-passo do funcionamento da máquina. Há diferenças na forma que Turing descreve as etapas: os estados são chamados de *m-config.* (abreviação de "*machine configuration*" ou "configuração de máquina"), as operações são chamadas de "*behaviour*" ("comportamento") e próximo estado é chamado de "*final m-config.*". Nas *skeleton tables* apresentadas por Turing aparecem letras alemãs maiúsculas e minúsculas e letras gregas minúsculas: as letras alemãs maiúsculas são usadas para definir cada configuração de máquina, as minúsculas são usadas para nomear funções e as letras gregas são usadas para definir variáveis, no entanto, apenas cinco letras maiúsculas alemãs são usadas por Turing (as cinco letras iniciais do alfabeto[11]).
Em todas as implementações do Turing, a escrita na fita segue um padrão: em uma sequência, os elementos estão separados por um espaço em branco e três espaços em branco de um elemento a outro separam duas sequências. A figura 4 ilustra um exemplo da sequência `01110`, os espaços extras servem para marcações, caso seja necessário realizar alguma contabilização, por exemplo. Turing identifica as sequências na fita como *F-squares*, ou seja, são

Figura 4: Exemplo da sequência `01110` escrita na fita.

espaços da fita que compõem uma sequência, uma *figura*, como ilustra a figura 4. Os outros espaços são chamados de *E-squares*, assim, espaços que serão reescritos, são espaços apagáveis (*erasable*).

[9] "*The skeleton tables are to be regarded as nothing but abbreviations: they are not essential. So long as the reader understands how to obtain the complete tables from the skeleton tables, there is no need to give any exact definitions in this connection*" (TURING, 1937b). Tradução do autor: As *skeleton tables* devem ser consideradas nada além de abreviações: elas não são essenciais. Assim que o leitor entende como obter as tabelas completas das *skeleton tables*, então não há necessidade de dar qualquer definição exata sobre isto.

[10] A máquina universal de Turing é uma máquina de Turing capaz de simular qualquer máquina, em outras palavras, pode se dizer que a máquina universal é programável - como descrito por Turing, é capaz de computar qualquer sequência computável. A máquina foi proposta e definida na sexta seção de *Computable* (TURING, 1937b).

[11] $\mathfrak{A}$ (A), $\mathfrak{B}$ (B), $\mathfrak{C}$ (C), $\mathfrak{D}$ (D) e $\mathfrak{E}$ (E)

A seguir, um exemplo de *Computable* (TURING, 1937b):

m-config.	*Symbol*	*Behaviour*	*Final m-config.*
$\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$	ə	L	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	not ə	L	$\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$
$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	None	R	$\mathfrak{f}_2(\mathfrak{C},\mathfrak{B},\alpha)$
$\mathfrak{f}_2(\mathfrak{C},\mathfrak{B},\alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	None	R	$\mathfrak{B}$

> From the *m*-configuration $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$ the machine finds the symbol of form α which is farthest to the left (the "first" α) and the *m*-configuration then becomes $\mathfrak{C}$. If there is no α then the *m*-configuration becomes $\mathfrak{B}$. [12]

Abaixo da tabela está a explicação dada por Turing. Essa *skeleton table* apresenta uma função que procura pelo α que está mais à esquerda na fita, assim, o primeiro α. A computação começa com a função $\mathfrak{f}$[13] que dependendo do que o cabeçote de fita estiver apontando, irá seguir um dos dois caminhos possíveis: se tiver um xevá (ə), o cabeçote move para a esquerda e a próxima configuração de máquina (ou o próximo estado) passa a ser $\mathfrak{f}_1$; se não tiver um xevá, o cabeçote move à esquerda e a máquina continua na configuração $\mathfrak{f}$, portanto, a máquina ficará neste estado, movendo o cabeçote à esquerda, em um *loop*, até achar o xevá.
Indo para $\mathfrak{f}_1$ há três possibilidades: se o cabeçote da fita está apontando para um α, indica que a máquina já achou o símbolo desejado e termina a computação, o cabeçote não se move e a máquina irá para a configuração final de aceitação $\mathfrak{C}$; se o cabeçote está apontando para algum símbolo que não é o α, então o cabeçote é movido para a direita e a configuração da máquina permanece em $\mathfrak{f}_1$; se o cabeçote está apontando para *nada* (*None*), ou seja, em um espaço vazio, o cabeçote move para a direita e a máquina passa para a configuração $\mathfrak{f}_2$.
Quando Turing menciona nas tabelas algo como "not α" ou "not ə" isso implica no caso do espaço ter um símbolo qualquer, não vazio, que não seja os respectivos.

[12] Tradução do autor: Da configuração de máquina $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$ a máquina encontra o símbolo da forma α que é o que está mais à esquerda (o "primeiro" α) e a configuração de máquina passa a ser $\mathfrak{C}$. Se não tem nenhum α então a configuração de máquina passa a ser $\mathfrak{B}$.

[13] Neste caso, o $\mathfrak{f}$ da *skeleton table* vem de "*find*" (encontrar).

No passo anterior, $\mathfrak{f}_1$, se o cabeçote está apontando para um espaço vazio, então vai para a configuração $\mathfrak{f}_2$. Nesta configuração há, também, três possibilidades: se o cabeçote apontar para um espaço com o símbolo α, então não há movimentos do cabeçote, pois já achou o símbolo desejado e a máquina vai para a configuração final $\mathfrak{C}$; se estiver apontando para algum símbolo qualquer não vazio que não é α, o cabeçote move-se para a direita e a configuração passa para $\mathfrak{f}_1$; por fim, se o cabeçote estiver apontando para *nada*, o cabeçote é movido para a direita e a máquina passa para a configuração final $\mathfrak{B}$, indicando que não achou nenhum α na fita, concluindo a computação.
Observando a função $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$ é notável os parâmetros extras referenciando os estados $\mathfrak{C}$ e $\mathfrak{B}$, apesar de ser possível referenciar a função apenas como $\mathfrak{f}(\alpha)$, Turing adiciona os parâmetros adicionais para fins de garantia que, eventualmente, a função $\mathfrak{f}$ irá parar num estado final, seja de caso afirmativo (achou o primeiro α, logo $\mathfrak{C}$) ou de caso negativo (não achou nenhum α, logo $\mathfrak{B}$). Uma forma de "ler" a função seria "a função $\mathfrak{f}$ procura pelo primeiro α na sequência, se a sequência tiver o α, então a máquina vai para o estado final $\mathfrak{C}$, se a sequência não tiver o α, então a máquina vai para o estado final $\mathfrak{B}$".
Agora, um outro exemplo de Turing em *Computable* (TURING, 1937b) que utiliza o exemplo anterior da função $\mathfrak{f}$:

$\mathfrak{e}(\mathfrak{C},\mathfrak{B},\alpha)$		$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C},\mathfrak{B},\alpha),\mathfrak{B},\alpha)$
$\mathfrak{e}_1(\mathfrak{C},\mathfrak{B},\alpha)$	E	$\mathfrak{C}$

> From $\mathfrak{e}(\mathfrak{C},\mathfrak{B},\alpha)$, the first α is erased and $\rightarrow \mathfrak{C}$. If there is no α $\rightarrow \mathfrak{B}$.[14]

Duas observações com o exemplo: primeiro é a tabela está formatada de forma diferente, porém, segue o mesmo estilo da tabela do exemplo anterior. É possível notar a ausência da coluna de símbolos, isto porque nesta *skeleton table* ela não é utilizada, logo, Turing decidiu removê-la; segunda observação é no texto explicativo ter uma seta ($\rightarrow$), segundo Turing, o símbolo é usado para significar "a máquina vai para a configuração de máquina ...".
Com esta tabela, a máquina começa na função $\mathfrak{e}$[15], que tem definido um α como entrada e dois estados finais, independente do símbolo que estiver na fita, a máquina não faz nenhuma operação e passa para o estado $\mathfrak{f}$. A função $\mathfrak{f}$ é usada, também tendo o α como entrada, porém, os dois primeiros parâmetros são um estado final $\mathfrak{B}$ já conhecido, como estado de rejeição, e onde costumava ficar o estado $\mathfrak{C}$ temos uma outra função $\mathfrak{e}_1$.
Um ponto importante que vale frisar: apesar de intuitivamente parecer que a função $\mathfrak{e}_1$, de alguma forma, vai ser computada antes da função $\mathfrak{f}$, este caso

[14] Tradução do autor: De $\mathfrak{e}(\mathfrak{C},\mathfrak{B},\alpha)$, o primeiro α é apagado e $\rightarrow \mathfrak{C}$. Se não tem $\alpha \rightarrow \mathfrak{B}$.
[15] Neste caso, o $\mathfrak{e}$ vem de "*erase*" (apagar).

não se aplica aqui. Como observado antes, a função $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ procura pelo primeiro α, se achar a máquina vai para $\mathfrak{C}$, caso contrário vai para $\mathfrak{B}$, em ambos os casos a máquina para de computar; é o comportamento similar em $\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$, só que ao achar o α a máquina vai para $\mathfrak{e}_1$ e a computação continua. A forma de uma função estar dentro da outra é chamada de função aninhada, estas que vão ser bastante usadas por Turing.
Seguindo com a computação: na função $\mathfrak{f}$ se a máquina não achar α, ela vai para o estado $\mathfrak{B}$ e a computação é terminada, por outro lado, se o α for encontrado, a máquina vai para o estado $\mathfrak{e}_1$. Por fim, na função $\mathfrak{e}_1$ e com o cabeçote apontando para α, o símbolo é apagado e a máquina vai para o estado final $\mathfrak{C}$, terminando a computação.
A seguir, mais um exemplo de *Computable* (TURING, 1937b), que usa a função $\mathfrak{e}$ anterior:

$$\mathfrak{e}(\mathfrak{B}, \alpha) \qquad \mathfrak{e}(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$$

From $\mathfrak{e}(\mathfrak{B}, \alpha)$ all letters α are erased and $\rightarrow \mathfrak{B}$.[16]

Na tabela vemos a definição de uma função também chamada $\mathfrak{e}$, porém, como um menor número de parâmetros e seu funcionamento é diferente da função $\mathfrak{e}$ do exemplo anterior, porém, ambas têm os mesmos objetivos que é apagar o α, logo, a função $\mathfrak{e}$ tem duas formas possíveis - em programação, chamamos isto de *sobrecarga de operador*. Outra observação é notar a função $\mathfrak{e}$ de dois parâmetros dentro da função de três parâmetros, duas funções aninhadas.
O comportamento da função $\mathfrak{e}$ de dois parâmetros é bem simples: ela chama a função de três parâmetros para apagar o α, ao achar e apagar o α, a função $\mathfrak{e}$ de três parâmetros chama a função de dois parâmetros, que repete a operação, até a função de três parâmetros não achar um α e ir para o estado $\mathfrak{B}$, terminando a computação. Aqui vemos um caso do uso de recursão, usar a função para chamar a si mesma, a fim de resolver tarefas repetidas (apagar α).
Assim, exibimos três tabelas de *Computable* que serão utilizadas posteriormente para a construção de novas, outras tabelas da publicação, que também serão úteis, estão descritas no apêndice A.
Seguindo com a seção de abreviações de (TURING, 1937a):

> A number of abbreviations of the same character as those in *Computable* (pp. 235-239) are introduced here. They will be applied in connection with the calculus of conversion, but are necessary for other purposes, e.g. for carrying out the processes of any ordinary formal logic with machines. The abbreviations in *Computable* are taken as known.

[16] Tradução do autor: De $\mathfrak{e}(\mathfrak{B}, \alpha)$ todas as letras α são apagadas e $\rightarrow \mathfrak{B}$.

> 'The sequence of symbols marked with α(followed by α)' will be abbreviated to $S(\alpha)$ in the explanations. Sequences are normally identified by the way they are marked, and are as it were lost when their marks are erased.
>
> In the tables $\mathfrak{B}$ will be used as a name for the symbol 'blank' (TURING, 1937a).[17]

Aqui são apresentadas novas funções que serão utilizadas para descrever o funcionamento de uma máquina de Turing capaz de computar Cálculo $\lambda - K$. As novas funções fazem o uso de funções já descritas em *Computable* (TURING, 1937b). Turing descreve que as novas funções, além de serem úteis para a computação de Cálculo $\lambda - K$, serão úteis para a máquina computar qualquer outro tipo de modelo formal. Uma abreviação que Turing deixa claro é sobre uma sequência de símbolos marcados com α, que será identificada como $S(\alpha)$ - lembrando que cada símbolo de uma sequência na fita tem um espaço extra entre si para marcações como apresentado na figura 4. O símbolo $\mathfrak{B}$ vai ser usado para descrever o símbolo vazio, ou seja, para marcar quando um espaço está em branco.
A partir de agora serão apresentadas as novas tabelas referentes às funções que serão necessárias para computar cálculo $\lambda - K$. A primeira nova função a ser definida por Turing, $\mathfrak{pem}$[18].

$\mathfrak{pem}(\mathfrak{A}, \alpha, \beta)$		$\mathfrak{pe}(\mathfrak{pem}_1, \alpha)$
$\mathfrak{pem}_1$	R, Pβ	$\mathfrak{A}$

> $\mathfrak{pem}_1$ here stands for $\mathfrak{pem}_1(\mathfrak{A}, \alpha, \beta)$ and similar abbreviations must be understood throughout.
> $\mathfrak{pem}(\mathfrak{A}, \alpha, \mathfrak{B})$. The machine prints α at the end of the sequence of symbols on F-squares and marks it with β. $\rightarrow \mathfrak{A}$.[19]

$\mathfrak{pem}$ faz uso da função $\mathfrak{pe}$ que Turing descreveu em *Computable*. $\mathfrak{pe}$ significa "*print at the end*" (imprima no final) e, seu funcionamento, é imprimir um

[17] Tradução do autor: Um número de abreviações de mesma forma das apresentadas em *Computable* (p. 235-293) são apresentadas aqui. Elas serão aplicadas em conexão com o cálculo de conversão, mas são necessárias para outros propósitos, e.g. para carregar o processo de qualquer lógica formal ordinária com máquinas. As abreviações de *Computable* são tomadas como conhecidas.

'A sequência de símbolos marcados com α(seguido por α)' será abreviado para $S(\alpha)$ nas explicações. Sequências são normalmente identificadas pela maneira que são marcadas e são como se tivessem perdidas quando seus marcadores são apagados.

Nas tabelas $\mathfrak{B}$ será usado para nomear o símbolo 'branco'.

[18] Neste caso, $\mathfrak{pem}$ significa "print at the end and mark" (imprima no final e marque).

[19] Tradução do autor: $\mathfrak{pem}_1$ significa $\mathfrak{pem}_1(\mathfrak{A}, \alpha, \beta)$ e abreviações similares devem ser entendidas da mesma forma.
$\mathfrak{pem}(\mathfrak{A}, \alpha, \mathfrak{B})$. A máquina escreve α no fim da sequência da símbolos de *F-squares* e marca-as com β. $\rightarrow \mathfrak{A}$.

elemento no final de uma sequência de *F-squares*, neste caso, imprime um α no final da sequência. As tabelas das funções que foram definidas em *Computable* foram transcritas no apêndice A, a tabela da função $\mathfrak{pe}$ está na seção A.1.
Uma observação notável é em relação à função $\mathfrak{pem}_1$ que compõe a função $\mathfrak{pem}$ e a forma abreviada que é usada na tabela, isto é apenas para facilitar a leitura das funções. $\mathfrak{pem}_1$ é chamada logo após $\mathfrak{pe}$ finalizar de executar, portanto, a máquina passou por uma sequência de *F-squares* e imprimiu α, o cabeçote ainda aponta para o α, para finalizar, o cabeçote é movido para a direita e é escrito β no espaço, indicando que a sequência foi marcada.
De forma similar à função $\mathfrak{pem}$, outras duas novas funções fazem uso de tabelas definidas em *Computable* e a tarefa extra de marcar os símbolos após escrevê-los, no caso, *copy and replace* ($\mathfrak{cr}$, apêncice A.6) e *copy and erase* ($\mathfrak{ce}$, apêndice A.4):

> The tables for $\mathfrak{crm}(\mathfrak{B},\gamma,\beta)$ and $\mathfrak{cem}(\mathfrak{B},\gamma,\beta)$ are to be obtained from those for $\mathfrak{cr}(\mathfrak{B},\gamma)$ and $\mathfrak{ce}(\mathfrak{B},\gamma)$ by replacing $\mathfrak{pe}(\mathfrak{A},\alpha)$ by $\mathfrak{pem}(\mathfrak{A},\alpha,\beta)$ throughout.[20]

Referente às funções $\mathfrak{cr}$ e $\mathfrak{ce}$ de dois parâmetros: a primeira função copia, na ordem da esquerda para a direita, todos os símbolos marcados com o símbolo especificado no segundo parâmetro da função, neste caso γ, e escreve-os no final da fita. A segunda função é similar à anterior, a única diferença é que os marcadores γ são apagados.

$\mathfrak{cem}(\mathfrak{B},\gamma,\beta)$	$\mathfrak{ce}(\mathfrak{pe}(\mathfrak{cem}(\mathfrak{B},\gamma,\beta),\beta),\mathfrak{B},\gamma)$
$\mathfrak{crm}(\mathfrak{B},\gamma,\beta)$	$\mathfrak{cr}(\mathfrak{pe}(\mathfrak{crm}(\mathfrak{B},\gamma,\beta),\beta),\mathfrak{B},\gamma)$

Tabela 4: As tabelas com as definições das funções $\mathfrak{cem}$ e $\mathfrak{crm}$.

$\mathfrak{crm}$ e $\mathfrak{cem}$, portanto, realizam as respectivas operações de $\mathfrak{cr}$ e $\mathfrak{ce}$, mas a sequência resultante é marcada com o símbolo especificado no terceiro parâmetro (β neste caso). As tabelas não são apresentadas por Turing, mas dá para deduzí-las, que foram reproduzidas na tabela 4.
A tabela a seguir apresenta um complemento de uma função definida em *Computable*, a função *compare*:

$\mathfrak{cpr}(\mathfrak{A},\mathfrak{C},\alpha,\beta)$	$\mathfrak{cp}(\mathfrak{cpr}_1,\mathfrak{cpr}_2,\mathfrak{cpr}_3,\alpha,\beta)$
$\mathfrak{cpr}_1$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{cpr},\mathfrak{b},\beta,b),\mathfrak{b},\alpha,a)$
$\mathfrak{cpr}_2$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{c},b,\beta),a,\alpha)$
$\mathfrak{cpr}_3$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{A},b,\beta),a,\alpha)$

[20]Tradução do autor: As tabelas de $\mathfrak{crm}(\mathfrak{B},\gamma,\beta)$ e $\mathfrak{cem}(\mathfrak{B},\gamma,\beta)$ são obtidas da mesma forma para $\mathfrak{cr}(\mathfrak{B},\gamma)$ e $\mathfrak{ce}(\mathfrak{B},\gamma)$ substituindo $\mathfrak{pe}(\mathfrak{A},\alpha)$ por $\mathfrak{pem}(\mathfrak{A},\alpha,\beta)$ em todas as ocasiões.

> $\mathfrak{cpr}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta)$. The machine compares $S(\alpha)$ with $S(\beta)$. $\rightarrow \mathfrak{A}$ if they are alike; $\rightarrow \mathfrak{C}$ otherwise. No erasures are made.
>
> The letters a, b occuring in the table for $\mathfrak{cpr}$ should not be used elsewhere in any machine whose table involves $\mathfrak{cpr}$. This can be made automatic by using $a_{\mathfrak{cpr}}$ and $b_{\mathfrak{cpr}}$, say, instead of a and b. We shall however write a and b and understand them to mean $a_{\mathfrak{cpr}}$ and $b_{\mathfrak{cpr}}$. The same applies for the letters a, ..., z in all such tables. [21]

A função $\mathfrak{cpr}$ compara se duas sequências marcadas com os respectivos símbolos nos terceiro e quarto parâmetros da função são iguais. Em *Computable*, a função compara dois símbolos marcados com os respectivos marcadores α e β (apêndice A.7). Seguindo por $\mathfrak{cp}$, há duas possibilidades de resultado da comparação das duas sequências. $\mathfrak{cpr}_1$ indica que o primeiro elemento de $S(\alpha)$ é igual ao primeiro elemento de $S(\beta)$, assim, o α é substituído por a e, em seguida na função interna de $\mathfrak{re}$, o β é substituído pelo b. A computação volta para $\mathfrak{cpr}$ que compara o segundo elemento das sequências, considerando que $S(\alpha)$ é igual a $S(\beta)$, este laço vai ser repetido até todos os α serem substituídos por a e todos os β serem substituídos por b. Não tendo mais comparações a fazer, o estado atual passa para $\mathfrak{cpr}_3$, a função $\mathfrak{re}$ externa substitui todos os a por α, em seguida, o $\mathfrak{re}$ interno substitui todos os b por β, desfazendo todas as marcações, e, por fim, a máquina vai para o estado $\mathfrak{A}$ terminando a computação.
$\mathfrak{cpr}_2$ indica que um elemento de $S(\alpha)$ não é igual a um elemento de $S(\beta)$, assim, as marcações são revertidas para os respectivos α e β e a máquina vai para o estado $\mathfrak{E}$.
Turing adicionou uma observação sobre os novos marcadores a e b, tais símbolos valem apenas para as respectivas tabelas, portanto, são símbolos internos da função, não sendo válidos fora dela. Quando outras tabelas também fizerem uso dos novos marcadores significa, portanto, que são símbolos usados para auxiliar na execução da função.
A próxima tabela foca mais na estrutura de uma função $\lambda - K$-definível, a função *brackets* marca os símbolos dentro dos colchetes mais externos, ou seja, procura pelos pares de colchetes:

---

[21] Tradução do autor: $\mathfrak{cpr}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta)$. A máquina compara $S(\alpha)$ com $S(\beta)$. $\rightarrow \mathfrak{A}$ se são iguais; $\rightarrow \mathfrak{C}$ caso contrário. Nenhuma operação de apagar é realizada.

As letras a, b que aparecem na tabela $\mathfrak{cpr}$ não devem ser usadas fora desta em qualquer máquina em qual a tabela envolve $\mathfrak{cpr}$. Isso pode ser feito automáticamente usando $a_{\mathfrak{cpr}}$ e $b_{\mathfrak{cpr}}$, digamos, ao invés de a e b. Nós iremos, no entanto, escrever a e b e entendê-los como $a_{\mathfrak{cpr}}$ e $b_{\mathfrak{cpr}}$. O mesmo se aplica para as letras a, ..., z em todas as respectivas tabelas.

$$
\begin{array}{rll}
\mathfrak{k}(\mathfrak{A},\gamma)\left\{\begin{array}{ll} \gamma & \text{L} \\ \text{not } \gamma & \text{R, R} \end{array}\right. & & \begin{array}{l} \mathfrak{A} \\ \mathfrak{k} \end{array} \\
\mathfrak{bk}(\mathfrak{A},\alpha,\beta,\gamma,\delta) & \text{E, P}a & \mathfrak{k}(\mathfrak{bk}_1,\gamma) \\
\mathfrak{bk}_1\left\{\begin{array}{ll} \alpha & \text{R, E, P}b \\ \beta & \text{L} \\ \text{others} & \text{R, R, R} \end{array}\right. & & \begin{array}{l} \mathfrak{k}(\mathfrak{bk}_2,\gamma) \\ \mathfrak{bk}_3 \\ \mathfrak{k}(\mathfrak{bk}_1,\gamma) \end{array} \\
\mathfrak{bk}_2\left\{\begin{array}{ll} \beta & \text{R, E, P}b \\ & \\ \text{not } \beta & \text{R, R, R} \end{array}\right. & & \begin{array}{l} \mathfrak{f}(\mathfrak{bk},\mathfrak{b},a) \\ \\ \mathfrak{k}(\mathfrak{bk}_2,\gamma) \end{array} \\
\mathfrak{bk}_3\left\{\begin{array}{ll} \gamma \text{ or } b & \text{E, P}\delta\text{, L, L} \\ \alpha & \text{E, P}\gamma \\ \text{others} & \text{L, L} \end{array}\right. & & \begin{array}{l} \mathfrak{bk}_3 \\ \mathfrak{A} \\ \mathfrak{bk}_3 \end{array}
\end{array}
$$

$\mathfrak{bk}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$. This describes the process of finding the partner of a bracket. If α and β are regarded as left and right brackets, then if the machine takes up the internal configuration $\mathfrak{bk}$ when scanning a square next on the right of an α it will find the partner of this α in the sequence $S(\gamma)$, and will mark the part of $S(\gamma)$ which is between the brackets with δ (instead of γ). The final internal configuration is $\mathfrak{A}$ and the scanned square is that which was scanned when the internal configuration $\mathfrak{bk}$ was first taken up. [22]

Analisando a definição em si da função *brackets*, $\mathfrak{bk}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$, podemos notar na ordem da esquerda para a direita, a configuração final de aceitação da máquina ($\mathfrak{A}$), o símbolo que representa um abre colchetes "[" (α), o símbolo que representa um fecha colchetes "]" (β), o marcador sendo usado na sequência de símbolos $S(\gamma)$ e o marcador que será usado dentro dos colchetes mais externos (δ). Usaremos a seguinte função de cálculo $\lambda - K$ para ilustrar o funcionamento da tabela $\mathfrak{bk}$: $\lambda x[\lambda x'[x']]$. A figura 5 ilustra a fita na configuração inicial que a máquina irá operar a função $\mathfrak{bk}$.

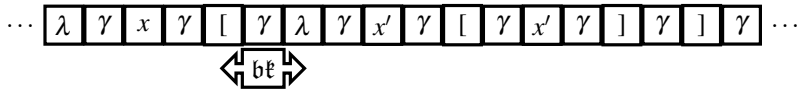


Figura 5: Conteúdo da fita com a máquina na configuração inicial.

[22] Tradução do autor: $\mathfrak{bk}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$. Esta descreve o processo de encontrar o par de um colchete. Se α e β são considerados como os colchetes esquerdo e direito, então se a máquina vai para a configuração interna $\mathfrak{bk}$ quando está lendo um espaço à direita de um α ela vai achar o par deste α na sequência $S(\gamma)$ e vai marcar a parte de $S(\gamma)$ que é entre os colchetes com δ (ao invés de γ). A configuração interna final é $\mathfrak{A}$ e o espaço lido é aquele que foi lido quando a configuração interna $\mathfrak{bk}$ foi inicialmente chamada.

Conforme especificado por Turing, o cabeçote começa apontando para o primeiro símbolo após a abertura do primeiro colchete, assim, na função $\mathfrak{bk}$ o símbolo apontado é substituído por a, um marcador local específico da tabela, em seguida, segue-se para a função $\mathfrak{k}$. A figura 6 ilustra o conteúdo da fita após a substituição do símbolo por a.

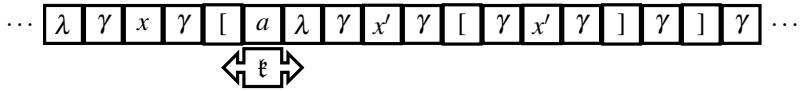


Figura 6: Conteúdo da fita com a máquina na configuração $\mathfrak{k}$.

Na função $\mathfrak{k}(\mathfrak{bk}_1, \gamma)$ existem duas condições, se o símbolo sendo apontado pelo cabeçote é ou não é γ, assim, procura pelo γ mais próximo seguindo-se à direita. O símbolo atual apontado não é γ, o cabeçote é movido duas vezes para a direita, esta operação é realizada até o primeiro γ ser encontrado. Após o γ ser encontrado, o cabeçote é movido para a esquerda e a computação passa para a função $\mathfrak{bk}_1$. A figura 7 ilustra o conteúdo da fita até então.

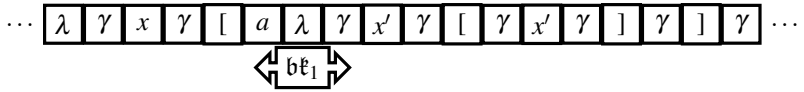


Figura 7: Conteúdo da fita com a máquina na configuração $\mathfrak{bk}_1$.

Na função $\mathfrak{bk}_1$ é feita a procura pelo primeiro α ou β, o cabeçote é movido três vezes para a direita até encontrar um dos dois símbolos mencionados, caso não ache α ou β, a função $\mathfrak{k}$ é usada para ajustar a procura. No exemplo, o cabeçote é movido três vezes para a direita, apontando para um γ, $\mathfrak{k}$ é chamado e faz o cabeçote ser movido para a esquerda, mais uma vez realizando as mesmas operações de $\mathfrak{bk}_1$ e $\mathfrak{k}$, mover três vezes para a direita e uma vez para a esquerda, o cabeçote passa a apontar para o primeiro α. O cabeçote é movido para a direita mais uma vez, o símbolo que está sendo apontado é substituido por b e novamente a função $\mathfrak{k}$ é usada. Após o próximo γ ser encontrado, segue-se para $\mathfrak{bk}_2$. A figura 8 mostra o conteúdo da fita até então.

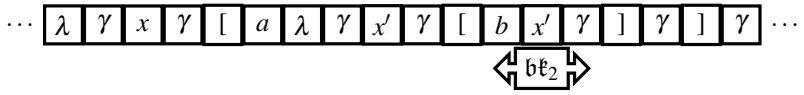


Figura 8: Conteúdo da fita com a máquina na configuração $\mathfrak{bk}_2$.

Na função $\mathfrak{bk}_2$ procura-se pelo β mais próximo. No exemplo, o símbolo apontado não é um β, então o cabeçote é movido três vezes para a direita,

$\mathfrak{k}$ verifica o γ mais próximo e move o cabeçote para a esquerda. Agora, a máquina está apontando para um β, o cabeçote é movido para a direita, o símbolo é substituído por b e a função $\mathfrak{f}(\mathfrak{bk}, \mathfrak{b}, a)$[23] é chamada, para procurar pelo primeiro a marcado e, ao encontrar, volta-se para $\mathfrak{bk}$. A figura 9 ilustra o conteúdo da fita até então.

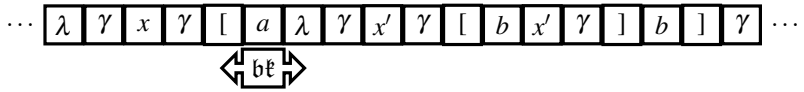


Figura 9: Conteúdo da fita com a máquina na configuração $\mathfrak{bk}$.

A função $\mathfrak{bk}$ chama $\mathfrak{k}$ para achar o γ mais próximo a direita, após isto, o cabeçote é movido para a esquerda e chama $\mathfrak{bk}_1$ para achar α ou β. Seguindo as operações de $\mathfrak{bk}_1$, o último β é encontrado, assim, o cabeçote é movido para a esquerda e segue-se para $\mathfrak{bk}_3$. A figura 10 mostra o conteúdo da fita até então.

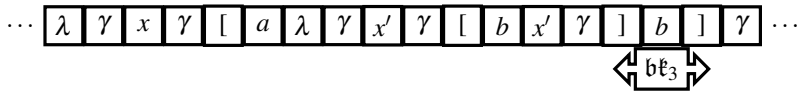


Figura 10: Conteúdo da fita com a máquina na configuração $\mathfrak{bk}_3$.

Por fim, em $\mathfrak{bk}_3$ todas as ocorrências de b ou γ são substituídas por δ até chegar ao marcador inicial a, logo, todos os símbolos que estão dentro dos colchetes mais externos estão marcados em γ, indicando o achamento dos dois colchetes. A figura 11 apresenta o conteúdo da fita.

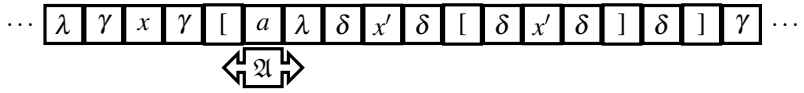


Figura 11: Conteúdo da fita com a máquina na configuração final $\mathfrak{A}$.

A tabela a seguir apresenta duas funções, $\mathfrak{sub}$ e $\mathfrak{dt}$ e ambas fazem uso da função $\mathfrak{sb}$.

[23] Observação para o $\mathfrak{b}$ para o caso da função não achar o primeiro a, isso é apenas um preenchimento que nunca será utilizado, pois não é possível chegar a $\mathfrak{bk}_2$ sem a máquina já ter escrito a em $\mathfrak{bk}_1$.

$\mathfrak{sb}(\mathfrak{A},\alpha,\beta,\gamma,\delta,\varepsilon)$			$\mathfrak{f}'(\mathfrak{sb}_1,\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{sb},a,j),b,\beta),\gamma,\varepsilon),\beta)$
$\mathfrak{sb}_1$	σ	R, E, Pb	$\mathfrak{sb}_2(\mathfrak{A},\alpha,\beta,\gamma,\delta,\varepsilon,\sigma)$
$\mathfrak{sb}_2$			$\mathfrak{f}'(\mathfrak{sb}_3,\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{A},b,\beta),j,\alpha),a,\alpha),a,\delta),\alpha)$
$\mathfrak{sb}_3$	σ	R, E, Pa	$\mathfrak{sb}$
	not σ	R, E, Pa	$\mathfrak{re}(\mathfrak{f}'(\mathfrak{sb}_4,\mathfrak{b},a),b,\beta)$
$\mathfrak{sb}_4$	τ	R, E, Pj	$\mathfrak{re}(\mathfrak{pem}(\mathfrak{sb},\tau,\delta),a,\alpha)$
$\mathfrak{sub}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$			$\mathfrak{sb}(\mathfrak{A},\alpha,\beta,\gamma,\delta,\delta)$
$\mathfrak{dt}(\mathfrak{A},\mathfrak{B},\alpha,\beta)$			$\mathfrak{pem}(\mathfrak{sb}(\mathfrak{f}(\mathfrak{e}(\mathfrak{A},d),\mathfrak{B},d),\alpha,\beta,p,\mathfrak{G},d),r,p)$

$\mathfrak{sub}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$. $S(\gamma)$ is substituted for $S(\beta)$ throughout $S(\alpha)$. The result is copied down and marked with δ. $\rightarrow \mathfrak{A}$. [24]

$\mathfrak{dt}(\mathfrak{A},\mathfrak{B},\alpha,\beta)$. It is determined whether the sequence $S(\beta)$ occurs in $S(\alpha)$. $\rightarrow \mathfrak{A}$ if it does; $\rightarrow \mathfrak{B}$ otherwise. [25]

A função $\mathfrak{sub}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$ resulta na *substituição* de $S(\gamma)$ por $S(\beta)$ em $S(\alpha)$, o resultado disto é marcado com δ, em outras palavras, é similar à aplicação da segunda regra de conversão direta de cálculo $\lambda - K$. Por exemplo, se $S(\alpha)$ for igual a `110`, $S(\beta)$ for igual a `1` e $S(\gamma)$ for igual a `011`, o resultado de $\mathfrak{sub}$ será $S(\delta)$ igual a `0110110`, assim, os `1` de $S(\alpha)$ foram substituídos por `011`.
A função $\mathfrak{dt}(\mathfrak{A},\mathfrak{B},\alpha,\beta)$ verifica se a sequência de símbolos marcados com β ocorre (está contido) na sequência marcada com α. Toda a execução da função é similar à da função $\mathfrak{sub}$, pois faz uso da função $\mathfrak{sb}$, no entanto duas observações são notáveis: quaisquer símbolos que em $\mathfrak{sub}$ seriam marcados com δ são ignorados (ao invés disso, são marcados com o símbolo vazio $\mathfrak{G}$) e o uso do marcador d que, se presente ao fim da execução, serve para indicar que a sequência $S(\beta)$ está contida em $S(\alpha)$.
As próximas três tabelas apresentadas serão relacionadas às possibilidades de resultados das funções lambda que serão utilizadas mais tarde, assim, relacionadas à enumeração.

> The tables which follow are particularly important in all cases where an enumeration of all possible results of operations of given types is required. The enumeration may be carried out by regarding the operation as determined by a number of choices, each between two possibilities, L and M say. Each possible sequence of operations is then associated with a finite sequence of letters L and M. These sequences can easily be enumerated. The method used here is to replace L by 0, each M by 1, follow the whole by 1, recerse the order and regard the result as the binary Arabic numeral corresponding to the given sequence. Thus the first few sequences (beginning with the one associated with 1)

[24] Tradução do autor: $\mathfrak{sub}(\mathfrak{A},\alpha,\beta,\gamma,\delta)$. $S(\gamma)$ é substituído por $S(\beta)$ por todo o $S(\alpha)$. O resultado é copiado e marcado com δ. $\rightarrow \mathfrak{A}$.

[25] Tradução do autor: $\mathfrak{dt}(\mathfrak{A},\mathfrak{B},\alpha,\beta)$. É determinado se a sequência $S(\beta)$ ocorre em $S(\alpha)$. $\rightarrow \mathfrak{A}$ se ocorre; $\rightarrow \mathfrak{B}$ caso contrário.

> are: the null sequence, L, M, LL, ML, LM, MM, LLL, MLL, LML, MML. In the general table below ζ and η are used instead of L and M. [26]

Em tabelas posteriores, vão existir casos em que uma decisão deve ser tomada e o resultado pode variar dependendo da decisão escolhida, no entanto, o *método* para chegar a cada resultado é diferente. Por exemplo, a figura 12 ilustra um grafo simples que sempre chega ao mesmo resultado F, porém, os caminhos possíveis para F podem variar: pode ser "*SadF*", "*SacF*", etc.. Se formos enumerar os resultados usando apenas os dois símbolos, teremos LM, LL, MM e assim por diante.

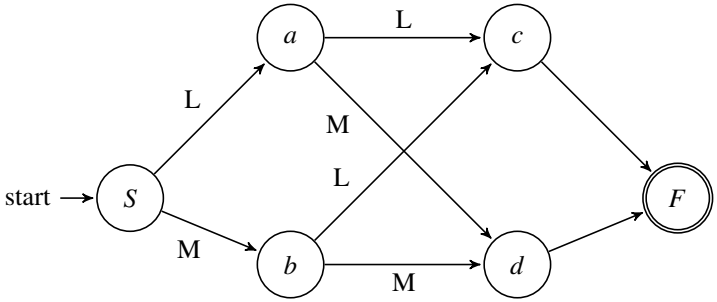


Figura 12: Exemplo de grafo onde o resultado é o mesmo, mas os caminhos para tal podem ser diferentes.

Assim, Turing define que todas as operações possíveis podem ser enumeradas usando apenas dois símbolos, L e M ou 0 e 1, ou seja, o uso de um sistema binário. Apesar da publicação de Turing ser pouco antes da tese de mestrado de Claude Elwood Shannon (1916-2001), que apresentou que o modelo de álgebra booleana poderia ser mapeado para um sistema binário, como também, o sistema binário poderia ser usado para descrever circuitos elétricos digitais (FILHO, 2007, Capítulo 5.7), Turing já tinha a justificativa do porquê usar um sistema binário ao invés do decimal para representar os resultados,

[26] As tabelas que se seguem são particularmente importantes em todos os casos onde uma enumeração de todos os possíveis resultados de operações de dados tipos é necessário. A enumeração pode ser realizada dependendo da operação como determinado por um número de escolhas, cada um entre duas possibilidades, L e M digamos. Cada sequência possível de operações é então associada à uma finita sequência de letras L e M. Essas sequências podem facilmente serem enumeradas. O método usado aqui é trocando L por 0, cada M por 1, seguindo o resto por 1, reverte a ordem e de acordo com o resultado como o numeral binário arábico correspondente à dada sequência. Portanto, as primeiras poucas sequências (começando com a sequência associada com 1) são: a sequência vazia, L, M, LL, ML, LM, MM, LLL, MLL, LML, MML. Na tabela geral abaixo ζ e η serão usados ao invés de L e M.

principalmente pela facilidade de lidar com apenas dois símbolos ao invés de dez:

> Trabalho binário é a coisa mais natural a se fazer em qualquer computador de larga escala. É muito mais fácil trabalhar na escala de dois do que qualquer outra, porque é tão fácil produzir mecanismos que tenham duas posições de estabilidade (PETZOLD, 2008, p.168, tradução nossa).[27]

A sequência de símbolos binários que Turing monta para a máquina não segue o modelo convencional de contagem de binário, ao invés disto, a sequência é feita de forma combinatória de acordo com o número de espaços disponíveis. Nas tabelas, ζ representará L ou 0 e η representará M ou 1.

$$\begin{array}{rlll} \mathfrak{add}(\mathfrak{A},\alpha,\zeta,\eta) & & & \mathfrak{f}'(\mathfrak{add}_1,\mathfrak{pem}(\mathfrak{add}_2,\zeta,\eta),\alpha) \\ \mathfrak{add}_1\Big\{ & \zeta & \text{R, E} & \mathfrak{pem}(\mathfrak{add},\zeta,\alpha) \\ & \eta & \text{R, E} & \mathfrak{pem}(\mathfrak{add}_2,\zeta,\alpha) \\ \mathfrak{add}_2 & & & \mathfrak{cem}(\mathfrak{re}(\mathfrak{A},a,\alpha),\alpha,a) \end{array}$$

$\mathfrak{add}(\mathfrak{A},\alpha,\zeta,\eta)$. The sequence $S(\alpha)$ consisting of letters ζ and η only is transformed into the next sequence. $\to \mathfrak{A}$. [28]

A função $\mathfrak{add}$ simplesmente adiciona mais um em uma sequência de símbolos marcados com α, assim, se o número na fita for `00`, o resultado de $\mathfrak{add}$ será `10`; se o número na fita for `110`, o resultado de $\mathfrak{add}$ será `001` e assim sucessivamente.

$$\begin{array}{rlll} \mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta) & & & \mathfrak{f}'(\mathfrak{ch}_1,\mathfrak{re}(\mathfrak{C},b,\alpha),\alpha) \\ \mathfrak{ch}_1\Big\{ & \zeta & \text{R, E, P}b & \mathfrak{A} \\ & \eta & \text{R, E, P}b & \mathfrak{B} \end{array}$$

$\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$ is an internal configuration which is taken up when a choice has to be made. $S(\alpha)$ is the sequence of letters ζ and η determining the choices. $\to \mathfrak{A}$ if the first unused letter is ζ; $\to \mathfrak{B}$ if it is η: it is then indicated that this ζ or η has been used by replacing its mark by b. When the whole sequence has been used up these marks are replaced again by α again and $\to \mathfrak{C}$. [29]

[27] Tradução livre do seguinte trecho: "*Binary working is the most natural thing to do with any large scale computer. It is much esier to work in the scale of two than any otherm because it is so easy to produce mechanisms which have two positions of stability.*".

[28] Tradução do autor: $\mathfrak{add}(\mathfrak{A},\alpha,\zeta,\eta)$. A sequência $S(\alpha)$ consistindo nas letras ζ e η apenas é transformada na próxima sequência. $\to \mathfrak{A}$.

[29] Tradução do autor: $\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$ é uma configuração interna que é chamada quando uma escolha tem de ser feita. $S(\alpha)$ é a sequência de letras ζ e η determinando as escolhas. $\to \mathfrak{A}$ se a primeira letra não usada é ζ; $\to \mathfrak{B}$ se for η; isso é então indicado que o ζ ou o η usado terá seu marcador substituído por b. Quando a sequência inteira for usada, os marcadores serão substituídos novamente por α e $\to \mathfrak{C}$.

A função $\mathfrak{ch}$ marca qual é o primeiro símbolo não usado de uma sequência $S(\alpha)$ e a configuração final passa a ser a de acordo: 0 (ζ) para a configuração final $\mathfrak{A}$ ou 1 (η) para a configuração final $\mathfrak{B}$. Cada vez que um símbolo da sequência de números já for usado, ele é marcado com b, se todos os símbolos forem marcados com b, então eles são desmarcados de volta para α e a configuração final da máquina passa a ser $\mathfrak{C}$.

$\mathfrak{cch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$		R	$\mathfrak{cch}_1$
$\mathfrak{cch}_1$	σ	E, Pa	$\mathfrak{cch}_2(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta,\sigma)$
$\mathfrak{cch}_2$			$\mathfrak{ch}(\mathfrak{f}(\mathfrak{cch}_3,\mathfrak{b},a),\mathfrak{f}(\mathfrak{cch}_4,\mathfrak{b},a),\mathfrak{C},\alpha,\zeta,\eta)$
$\mathfrak{cch}_3$		E, Pσ, L	$\mathfrak{A}$
$\mathfrak{cch}_4$		E, Pσ, L	$\mathfrak{B}$

$\mathfrak{cch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$. This differs from $\mathfrak{ch}$ in that the internal configurations $\mathfrak{A}$ and $\mathfrak{B}$ are taken up when the same square is scanned as that which was scanned when the internal configuration $\mathfrak{cch}$ was first reached, provided that this was an F-square. [30]

Na função $\mathfrak{cch}$, o cabeçote está apontando para um símbolo binário, após a verificação ter sido feita com $\mathfrak{ch}$, o cabeçote volta a apontar para o mesmo símbolo binário em que estava antes de chamar a função $\mathfrak{cch}$. O símbolo σ indica o marcador do número que estava sendo apontado inicialmente, que é substituído temporariamente por a que serve como sinalizador para a função $\mathfrak{cch}_2$: após a verficação em $\mathfrak{ch}$, o cabeçote volta para o a e desfaz a substituição.

3.4 CONVERSÃO MECÂNICA

Na seção anterior, apresentamos as tabelas das funções com operações programadas para uma Máquina de Turing realizar. A seção a seguir, usando as funções especificadas anteriormente, iremos abordar a forma que a máquina realiza as conversões de cálculo $\lambda - K$, os três casos possíveis de conversão direta de fórmulas bem formadas, que foram apresentadas em 3.2. As conversões são as operações de transformação em cálculo $\lambda - K$, assim, com elas que será possível apresentar a computabilidade das funções $\lambda - K$-definíveis. A primeira conversão a ser definida é a (iii) da lista 3.2, mas antes disso, Turing apresenta três funções que serão utilizadas nas funções das conversões: $\mathfrak{funk}$, $\mathfrak{ch}$ e $\mathfrak{cch}$, sendo as duas últimas complemento de funções já definidas.

[30] Tradução do autor: $\mathfrak{cch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\alpha,\zeta,\eta)$. Este diferencia de $\mathfrak{ch}$ em que as configurações internas $\mathfrak{A}$ e $\mathfrak{B}$ são retomadas quando o mesmo espaço é lido como aquele que foi lido quando a configuração interna $\mathfrak{cch}$ foi chamada pela primeira vez, desde que isso era uma *F-square*.

$\mathfrak{funk}(\mathfrak{A},\alpha,\beta,\gamma)$ $\mathfrak{pem}(\mathfrak{crm}(\mathfrak{pem}_2(\mathfrak{crm}(\mathfrak{pem}(\mathfrak{A},],\gamma),\beta,\gamma),],[,\gamma),\alpha,\gamma),[,\gamma)$

$\mathfrak{funk}(\mathfrak{A},\alpha,\beta,\gamma)$. $[S(\alpha)][S(\beta)]$ is written at the end and marked with γ. $\rightarrow \mathfrak{A}$. [31]

A função $\mathfrak{funk}$ simplesmente coloca duas sequências marcadas com símbolos α e β no final da fita e em colchetes e marca-as com γ. A escrita é feita da esquerda para a direita: o símbolo de abre colchetes é escrito no final da fita, marcado com γ, em seguida a sequência de símbolos marcados com α são escritos no final da fita e, cada um, marcado com γ. A função utilizada neste próximo passo, $\mathfrak{pem}_2(\mathfrak{A},\alpha,\beta,\gamma)$, não foi definida antes, no entanto, é possível notar que é uma extensão de $\mathfrak{pem}_1(\mathfrak{A},\alpha,\beta)$, imprime dois símbolos no final e os marca com γ, então podemos presumir a $\mathfrak{pem}_2$, descrita na tabela 5. Turing já havia apresentado casos de extensões de funções em *Computable*, por exemplo, com as funções *print at the end* e *copy and erase*, apresentadas nos apêndices A.1 e A.4, respectivamente.

$\mathfrak{pem}_2(\mathfrak{A},\alpha,\beta,\gamma)$	$\mathfrak{pem}(\mathfrak{pem}(\mathfrak{A},\beta,\gamma),\alpha,\gamma)$

Tabela 5: A tabela com a definição da função $\mathfrak{pem}_2$.

Após $\mathfrak{pem}_2$ o conteúdo escrito no final da fita será, desconsiderando os marcadores γ, "$[S(\alpha)][$". Para finalizar, as duas últimas funções transcrevem os símbolos marcados com β para o final da fita, marcando-os com γ e escreve o símbolo de fechar colchetes, também marcando-o com γ, respectivamente. O resultado final na fita é ilustrado na figura 13, onde a sequência `100` é marcada com α e `1` é marcada com β.

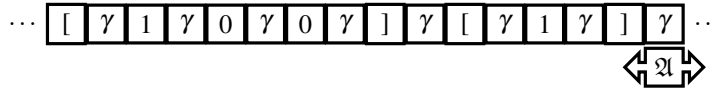


Figura 13: Saída da função $\mathfrak{funk}$ escrita no final da fita.

$\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\theta)$ $\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\theta,L,M)$
$\mathfrak{cch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\theta)$ $\mathfrak{ch}(\mathfrak{A},\mathfrak{B},\mathfrak{C},\theta,L,M)$

The choices will be determined by a sequence made up of letters L and M. [32]

[31] Tradução do autor: $\mathfrak{funk}(\mathfrak{A},\alpha,\beta,\gamma)$. É escrito $[S(\alpha)][S(\beta)]$ no fim e é marcado com γ. $\rightarrow \mathfrak{A}$.

[32] Tradução do autor: As escolhas serão determinadas por uma sequência composta de letras L e M.

As funções $\mathfrak{ch}$ e $\mathfrak{cch}$ de quatro parâmetros são similares às respectivas funções de seis parâmetros, o diferencial é a não necessidade de especificar os dois símbolos ζ e η a serem escolhidos, por padrão já é considerado que tais símbolos serão uma sequência de L e/ou M, marcados com θ.

$\mathfrak{pff}(\mathfrak{A},\mathfrak{C},\alpha,\theta)$			$\mathfrak{pe}_5(\mathfrak{c}(\mathfrak{h}(\mathfrak{af},\mathfrak{pff}_2,\mathfrak{C},\theta),:,;,x,;,x)$
$\mathfrak{pff}_1$			$\mathfrak{q}(\mathfrak{pff}_3,:)$
$\mathfrak{pff}_2$	;	R, R	$\mathfrak{cch}(\mathfrak{pff}_3,\mathfrak{pff}_2,\mathfrak{C},\theta)$
	$\mathfrak{B}$		$\mathfrak{af}$
	others	R, R	$\mathfrak{pff}_2$
$\mathfrak{pff}_3$	;		$\mathfrak{pff}_4$
	$\mathfrak{B}$		$\mathfrak{af}$
	others	R, Pa, R	$\mathfrak{pff}_3$
$\mathfrak{pff}_4$	;	R,R	$\mathfrak{cch}(\mathfrak{pff}_5,\mathfrak{pff}_4,\mathfrak{C},\theta)$
	$\mathfrak{B}$		$\mathfrak{af}$
	others	R, R	$\mathfrak{pff}_4$
$\mathfrak{pff}_5$	; or $\mathfrak{B}$		$\mathfrak{ar}$
	others	R, Pb, R	$\mathfrak{pff}_5$
$\mathfrak{ar}$			$\mathfrak{ch}(\mathfrak{ne},\mathfrak{ch}(\mathfrak{comp},\mathfrak{ab},\mathfrak{C},\theta),\mathfrak{C},\theta)$
$\mathfrak{ne}$			$\mathfrak{pe}_2(\mathfrak{ne}_1,;,x)$
$\mathfrak{ne}_1$			$\mathfrak{ch}(\mathfrak{pe}(\mathfrak{ne}_1,'),\mathfrak{af},\mathfrak{C},\theta)$
$\mathfrak{comp}$			$\mathfrak{pe}(\mathfrak{funk}(\mathfrak{af},a,b,\mathfrak{B}),;)$
$\mathfrak{ab}$			$\mathfrak{pe}_3(\mathfrak{ab}_1,;,\lambda,x)$
$\mathfrak{ab}_1$			$\mathfrak{ch}(\mathfrak{pe}(\mathfrak{ab}_1,'),\mathfrak{ab}_2,\mathfrak{C},\theta)$
$\mathfrak{ab}_2$			$\mathfrak{pe}(\mathfrak{ce}(\mathfrak{pe}(\mathfrak{af},]),a),[)$
$\mathfrak{af}$			$\mathfrak{e}(\mathfrak{e}(\mathfrak{ch}(\mathfrak{fin},\mathfrak{pff}_1,\mathfrak{C},\theta),a),b)$
$\mathfrak{fin}$			$\mathfrak{q}(\mathfrak{r}(\mathfrak{r}(\mathfrak{fin}_1)),;)$
$\mathfrak{fin}_1$	not $\mathfrak{B}$	R, Pα, R	$\mathfrak{fin}_1$
	$\mathfrak{B}$		$\mathfrak{A}$

$\mathfrak{pff}(\mathfrak{A},\mathfrak{C},\alpha,\theta)$[33]. A properly-formed formula is chosen, writen down at the end and marked with α. $\rightarrow\mathfrak{A}$. This is done by writing down successive properly-formed formulae separated by semicolons, and obtaining others from them by abstraction (i.e., the process by which $\lambda V[M]$ is obtained from M), by application of a function to its argument (i.e., obtaining $[M][N]$ from M and N), and by writing down new variables. Before writing down a new formula we have the alternative of taking the last formula as the result of the calculation. In this case the internal configuration $\mathfrak{fin}$ is taken up. If a new formula is to be constructed then two of the old formulae are chosen and marked with a and b: then one of the internal configurations $\mathfrak{ab}$, $\mathfrak{comp}$, $\mathfrak{ne}$ is chosen and the new

> formula is correspondingly $\lambda V[S(a)]$, $[S(a)][S(b)]$, or V, where V is a new variable. The whole of the work is separated by a colon from the symbols which were on the tape previously. The meanings of $\mathfrak{pe}_3$ and $\mathfrak{pe}_5$ are analogous to $\mathfrak{pe}_2$. [34]

A função $\mathfrak{pff}$ trata da terceira regra de conversão direta de cálculo $\lambda - K$. A realização das transformações é feita considerando duas observações pré-estabelecidas: a primeira é de uma sequência de símbolos binários L e M marcados com θ, indicando qual operação deve ser escolhida quando um $\mathfrak{ch}$ ou um $\mathfrak{cch}$ estiver sendo executado; a segunda é a consideração de que as fórmulas bem formadas a serem trabalhadas já estão escritas na fita e separadas por ponto e vírgula. O símbolo de dois pontos separa o que então está escrito na fita com o que será realizado em $\mathfrak{pff}$, após o símbolo de dois pontos, as fórmulas bem formadas a serem transformadas serão marcadas com a e, se necessário, b e isto é feito durante a execução das funções $\mathfrak{pff}_3$ e $\mathfrak{pff}_5$, respectivamente. Nota-se que antes de uma das funções ser chamada, há a escolha $\mathfrak{cch}$ para ir para a função seguinte para marcar a fórmula ou continuar movendo o cabeçote para frente até o próximo ponto e vírgula. Após as fórmulas serem marcadas, a configuração atual é a função $\mathfrak{ar}$ que, novamente se baseando na sequência marcada com θ define qual a transformação a ser realizada de acordo com os símbolos marcados, há três possibilidades: a função $\mathfrak{ne}$ é para a criação de uma nova variável (x, x', x'', etc.); a função $\mathfrak{comp}$ transforma os símbolos marcados com a e b para $[S(a)][S(b)]$; a função $\mathfrak{ab}$ converte os símbolos marcados com a com a definição lambda ($S(a)$ *conv* $\lambda x[S(a)]$).

Após uma das três possibilidades listadas anteriormente ser executada, na função $\mathfrak{af}$ há mais uma decisão, voltar o cabeçote para o começo da sequência de fórmulas, partindo de $\mathfrak{pff}_1$, para mais transformações ou terminar, para o último caso, a função $\mathfrak{fin}$ marca a última fórmula da lista com α como resultado da computação da função $\mathfrak{pff}$.

---

[34] Tradução do autor: $\mathfrak{pff}(\mathfrak{A}, \mathfrak{C}, \alpha, \theta)$. Uma fórmula bem formada é escolhida, escrita no fim e marcada com α. $\rightarrow \mathfrak{A}$. Isso é feito escrevendo sucessivas fórmulas bem formadas separadas por ponto e vírgula e obtendo outras dessas por abstração (i.e., o processo de $\lambda V[M]$ é obtido de M), pela aplicação de uma função ao seu argumento (i.e., obtendo $[M][N]$ de M e N), e escrevendo novas variáveis. Antes de escrever uma nova fórmula, nós temos a alternativa de pegar a última fórmula como resultado do cálculo. Neste caso a configuração interna $\mathfrak{fin}$ é retomada. Se uma nova fórmula será construída, então duas das fórmulas antigas são escolhidas e marcadas com a e b: então uma das configurações internas $\mathfrak{ab}$, $\mathfrak{comp}$, $\mathfrak{ne}$ é escolhida e a nova fórmula é correspondente a $\lambda V[S(a)]$, $[S(a)][S(b)]$ ou V, onde V é uma nova variável. Todo o trabalho é separado por uma vírgula dos símbolos nos quais já estavam na fita antes. Os significados de $\mathfrak{pe}_3$ e $\mathfrak{pe}_5$ são análogos ao significado de $\mathfrak{pe}_2$.

[34] É possível notar um erro de digitação cometido por Turing nesta tabela, na primeira linha à direita há um parêntese extra na função $\mathfrak{pe}_5$, no meio do nome da função $\mathfrak{ch}$.

A função $\mathfrak{va}$, a seguir, trata da primeira regra da conversão direta do cálculo $\lambda - K$, item (i) da lista 3.2:

$\mathfrak{va}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$			$\mathfrak{f}'(\mathfrak{va}_1,\mathfrak{A},\alpha)$
$\mathfrak{va}_1$	λ	R, E, Pa	$\mathfrak{f}'(\mathfrak{va}_2,\mathfrak{b},\alpha)$
	others		$\mathfrak{crm}(\mathfrak{A},\alpha,\beta)$
$\mathfrak{va}_2$	x or $'$	R, E, Pb	$\mathfrak{f}'(\mathfrak{va}_2,\mathfrak{b},\alpha)$
	others	R	$\mathfrak{bk}(\mathfrak{pe}(\mathfrak{va}_3,x),[,],\alpha,c)$
$\mathfrak{va}_3$		R, Pd	$\mathfrak{ch}(\mathfrak{pe}(\mathfrak{va}_3,'),\mathfrak{dt}(\mathfrak{va}_4,\mathfrak{va}_5,c,d),\mathfrak{C},\theta)$
$\mathfrak{va}_4$			$\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{crm}(\mathfrak{e}(\mathfrak{A},d),\alpha,\beta),b,\alpha),a,\alpha),c,\alpha)$
$\mathfrak{va}_5$			$\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{va}_6,a,\alpha),b,\alpha),c,\alpha),b,f)$
$\mathfrak{va}_6$			$\mathfrak{sub}(\mathfrak{e}(\mathfrak{e}(\mathfrak{A},d),f),\alpha,f,d,\beta)$

> $\mathfrak{va}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$. An immediate transformation (i) is chosen, and if permissible is carried out on $S(\alpha)$, the result being marked with β. If the chosen transformation is not permissible then $S(\beta)$ is identical with $S(\alpha)$. $\rightarrow \mathfrak{A}$. [35]

Conforme descrito por Turing: a entrada corresponde aos símbolos marcados com α e o resultado será marcado com β, novamente, considera-se uma sequência de símbolos marcados com θ que enumera as operações desejadas. Primeiramente, é procurado pelo primeiro α na sequência e o cabeçote é movido para a esquerda, indo para $\mathfrak{va}_1$. Em $\mathfrak{va}_1$ se o símbolo apontado pelo cabeçote não é um λ, então não é permitido realizar a transformação direta (i), os símbolos marcados com α são copiados para o final da sequência e marcados como β, terminando a computação; se o símbolo apontado for um λ, então este é marcado com a e procura-se pelo próximo α na sequência, indo para $\mathfrak{va}_2$.

Em $\mathfrak{va}_2$ as variáveis de entrada (x, x', x'', etc.) são marcadas com b, quando não houver mais variáveis, ou seja, o cabeçote apontar para um símbolo de abre colchetes, o cabeçote move-se para a direita e a função $\mathfrak{bk}$ é usada. Como já definido antes na seção 3.3, aqui a função $\mathfrak{bk}$ irá marcar com c os símbolos dentro dos colchetes e, após, irá imprimir x no fim da sequência.

Em $\mathfrak{va}_3$ o x impresso no fim da sequência é marcado com d, em seguida há dois caminhos possíveis e será decidido conforme o primeiro símbolo binário marcado com θ especificar. Se for L ou 0, então um símbolo $'$ será impresso e volta para $\mathfrak{va}_3$ que irá marcar tal símbolo com d, o x que até então estava escrito no final, agora é x' e toda a vez que a sequência θ for L, mais $'$ serão adicionados no x; se o primeiro símbolo marcado com θ for M ou 1, então há uma comparação, usando a função $\mathfrak{dt}$: se os símbolos marcados com d (símbolos no fim da sequência) for subsequência dos símbolos marcados com

[35] Tradução do autor: $\mathfrak{va}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$. Uma transformação imediata (i) é escolhida e se possível é realizada em $S(\alpha)$, o resultado é marcado com β. Se a transformação escolhida não é permitida então $S(\beta)$ é igual a $S(\alpha)$. $\rightarrow \mathfrak{A}$.

c (símbolos que estão dentro dos colchetes), então irá para a função $\mathfrak{va}_4$, caso contrario, irá para $\mathfrak{va}_5$.
Em $\mathfrak{va}_4$ indica que a subsequência d é a mesma, ou seja, é possível obter d de c, assim, a transformação é permitida. Todos os marcadores que foram escritos durante a execução da função $\mathfrak{va}$ voltam para α e, por fim, a sequência marcada com α é copiada para o final da sequência e marcada com β e os símbolos marcados com d são apagados e a computação é terminada.
Em $\mathfrak{va}_5$ os símbolos marcados com b (as variáveis de entrada, os símbolos que vêm logo depois do λ e antes do símbolo de abre colchetes) são copiados para o fim da sequência e marcados com f e todos os símbolos que foram escritos voltam para α, seguindo para $\mathfrak{va}_6$. A transformação não foi permitida, assim, em $\mathfrak{va}_6$ a sequência de símbolos marcados com f que está em α é copiada para o final da sequência e marcada com β, por fim, os símbolos marcados com f e d são apagados.
A seguir, a função $\mathfrak{red}(\mathfrak{A},\alpha,\beta)$ realiza a segunda regra de transformação direta, item (ii) da lista 3.2, considerando que a entrada da função está marcada com símbolos α e a saída da função marcada com β.

$\mathfrak{red}(\mathfrak{A},\alpha,\beta)$			$\mathfrak{f}'(\mathfrak{red}_1,\mathfrak{red}_{13},\alpha)$
$\mathfrak{red}_1$	[	R	$\mathfrak{bk}(\mathfrak{red}_2,[,],a,c)$
	not [		$\mathfrak{red}_{13}$
$\mathfrak{red}_2$		E, Pf	$\mathfrak{re}(\mathfrak{f}(\mathfrak{red}_3,\mathfrak{b},\alpha),\mathfrak{b},\alpha,f)$
$\mathfrak{red}_3$			$\mathfrak{bk}(\mathfrak{red}_4,[,],\alpha,d)$
$\mathfrak{red}_4$			$\mathfrak{f}'(\mathfrak{red}_5,\mathfrak{b},c)$
$\mathfrak{red}_5$	λ	R, E, Pf	$\mathfrak{f}'(\mathfrak{red}_6,\mathfrak{b},c)$
	not λ		$\mathfrak{red}_{13}$
$\mathfrak{red}_6$	x or $'$	R, E, Pg	$\mathfrak{f}'(\mathfrak{red}_6,\mathfrak{b},c)$
	[	R, E, Pf	$\mathfrak{q}(\mathfrak{red}_7,c)$
$\mathfrak{red}_7$		E, Pf	$\mathfrak{red}_8$
$\mathfrak{red}_8$			$\mathfrak{f}'(\mathfrak{red}_9,\mathfrak{red}_{10},c)$
$\mathfrak{red}_9$	λ	R, E, Pk	$\mathfrak{f}'(\mathfrak{red}_{11},\mathfrak{b},c)$
	not λ	R, E, Pk	$\mathfrak{red}_8$
$\mathfrak{red}_{11}$	x or $'$	R, E, Pj	$\mathfrak{f}'(\mathfrak{red}_{11},\mathfrak{b},c)$
	[		$\mathfrak{cpr}(\mathfrak{red}_{13},\mathfrak{red}_{12},j,g)$
$\mathfrak{red}_{12}$			$\mathfrak{dt}(\mathfrak{red}_{13},\mathfrak{re}(\mathfrak{red}_8,j,k),d,j)$
$\mathfrak{red}_{10}$			$\mathfrak{sub}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{A},d,\alpha),f,\alpha),k,\alpha),g,\alpha)k,g,d,\beta)))$
$\mathfrak{red}_{13}$			$\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{re}(\mathfrak{crm}(\mathfrak{A},\alpha,\beta),d,\alpha),g,\alpha),c,\alpha),f,\alpha),k,\alpha),j,\alpha)$

$\mathfrak{red}(\mathfrak{A},\alpha,\beta)$[36]. An immediate transformation (ii) is carried out on $S(\alpha)$, supposing that $S(\alpha)$ is properly-formed. The result is marked with β. $\rightarrow \mathfrak{A}$. If the transformation is not possible or permissible $S(\beta)$ is identical with $S(\alpha)$. Considerable use is made of the hypothesis that $S(\alpha)$ is properly-formed. Thus if its first symbol is [then it must be of form $[L][N]$ and if in addition the second symbol is λ then it is of form $[\lambda V[M]][N]$. The internal configuration $\mathfrak{red}_8$ is never reached unless $S(\alpha)$ is of this form, and in that case it first occurs when V has been marked with g, M with c, and N with d, the remaining symbols of what was $S(\alpha)$ being now marked with α or f. It is then determined whether the

> immediate transformation (ii) is permissible: if it is then $\mathfrak{red}_{10}$ is taken up and the substitution carried out. [37]

A função segue, estritamente, a forma $[\lambda V[M][N]]$, onde V é a variável de entrada e M e N são fórmulas bem definidas. No caso de não ser possível realizar a transformação, a saída $S(\beta)$ é igual a $S(\alpha)$ (chamado na função $\mathfrak{red}_{13}$). Em resumo, a função $\mathfrak{red}$ ajusta os marcadores de acordo, assim, em $[\lambda V[M][N]]$, V será marcado com g, M será marcado com c e N será marcado com d, por fim, a função $\mathfrak{sub}$, que está em $\mathfrak{red}_{10}$, é usada para fazer a transformação, ou seja, aplicar N em V por M, onde o resultado será escrito no final da sequência e marcado com β. Por exemplo, se V for igual a x, M for igual a x e N for igual a x' o resultado da transformação será x' ($[\lambda x[x]][x']$ *conv* x').

Por fim, Turing define mais duas funções que farão uso das três conversões já definidas: $\mathfrak{imc}$, referente a uma conversão imediata de uma função $\lambda - K$; $\mathfrak{conv}$, referente a uma série de conversões imediatas de uma função $\lambda - K$ até resultar na forma normal.

$$
\begin{array}{rll}
\mathfrak{imc}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta) & & \mathfrak{ch}(\mathfrak{f}'(\mathfrak{imc}_1,\mathfrak{imc}_2,\alpha),\mathfrak{re}(\mathfrak{imc}_1,\alpha,a),\mathfrak{C},\theta) \\
\mathfrak{imc}_1 \left\{ \begin{array}{ll} [& \text{R, E, P}c \\ \text{not } [& \text{R, E, P}c \end{array} \right. & \mathfrak{ch}(\mathfrak{imc},\mathfrak{imc}_3,\mathfrak{C},\theta) \\
 & & \mathfrak{imc} \\
\mathfrak{imc}_2 & & \mathfrak{re}(\mathfrak{crm}(\mathfrak{A},\alpha,\beta),c,\alpha) \\
\mathfrak{imc}_3 & & \mathfrak{q}(\mathfrak{bk}(\mathfrak{imc}_4,[,],\alpha,a),c) \\
\mathfrak{imc}_4 & & \mathfrak{ch}(\mathfrak{dc},\mathfrak{ch}(\mathfrak{rc},\mathfrak{er},\mathfrak{C},\theta),\mathfrak{C},\theta) \\
\mathfrak{dc} & & \mathfrak{va}(\mathfrak{imc}_5,\mathfrak{C},a,b,\theta) \\
\mathfrak{rc} & & \mathfrak{rev}(\mathfrak{imc}_5,a,b) \\
\mathfrak{er} & & \mathfrak{pff}(\mathfrak{rev}(\mathfrak{er}_1,b,d),\mathfrak{C},b,\theta) \\
\mathfrak{er}_1 & & \mathfrak{cpr}(\mathfrak{e}(\mathfrak{imc}_5,d),\mathfrak{e}(\mathfrak{e}(\mathfrak{crm}(\mathfrak{imc}_5,a,b),d),b),d,a) \\
\mathfrak{imc}_5 & & \mathfrak{crm}(\mathfrak{crm}(\mathfrak{crm}(\mathfrak{re}(\mathfrak{re}(\mathfrak{e}(\mathfrak{A},b),c,\alpha),a,\alpha),\alpha,\beta),b,\beta),c,\beta)
\end{array}
$$

> $\mathfrak{imc}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$. An immediate conversion is chosen and performed on $S(\alpha)$. The result is marked with β. $\rightarrow \mathfrak{A}$. [38]

A função $\mathfrak{imc}$ realiza uma transformação imediata numa função $\lambda - K$ marcadas com α, caso a transformação não for possível, pela função $\mathfrak{imc}_2$ os símbolos marcados com α são copiados para o fim da sequência e marcados

[37] Tradução do autor: $\mathfrak{red}(\mathfrak{A},\alpha,\beta)$. Uma transformação imediata (ii) é realizada em $S(\alpha)$, supondo que $S(\alpha)$ é bem formado. O resultado é marcado com β. $\rightarrow \mathfrak{A}$. Se a transformação não for possível ou permitida $S(\beta)$ é idêntico a $S(\alpha)$. Uso considerável é feito na hipótese de que $S(\alpha)$ é bem formado. Disto se o primeiro símbolo é [então precisa ser da forma $[L][N]$ e se em adição ao segundo símbolo é λ então é da forma $[\lambda V[M]][N]$. A configuração interna $\mathfrak{red}_8$ nunca é alcançada a menos que $S(\alpha)$ é desta forma, e neste caso ele ocorre primeiramente quando V foi marcado com g, M com c e N com d, os símbolos restantes do que era $S(\alpha)$ é então marcado com α ou f. É então determinado se a transformação imediata (ii) é permitida: se é, então $\mathfrak{red}_{10}$ é chamado e a substituição é realizada.

[37] Há um erro de digitação cometido por Turing nesta função. Na função $\mathfrak{sub}$ em $\mathfrak{red}_{10}$ falta uma vírgula após a definição do primeiro parâmetro (as funções $\mathfrak{re}$ aninhadas).

[38] Tradução do autor: $\mathfrak{imc}(\mathfrak{A},\mathfrak{C},\alpha,\beta,\theta)$. Uma conversão imediata é escolhida e realizada em $S(\alpha)$. O resultado é marcado com β. $\rightarrow \mathfrak{A}$.

com β. Novamente, a função $\mathfrak{imc}$ utiliza de uma sequência de símbolos marcados com θ para definir qual operação deve ser utilizada em dada ocasião.
A primeira função a ser executada é uma decisão: procurar pelo primeiro α na sequência e mover o cabeçote para a esquerda, tendo α, segue para $\mathfrak{imc}_1$, caso contrário, irá para $\mathfrak{imc}_2$; substituir todos os α da sequência por a e seguir para $\mathfrak{imc}_1$.
Em $\mathfrak{imc}_1$ o marcador do próximo símbolo é substituído por c, no entanto, se o símbolo a ser marcado for um de abre colchetes, há a escolha de voltar para $\mathfrak{imc}$ ou seguir para $\mathfrak{imc}_3$; se o símbolo a ser marcado for qualquer outro, volta para $\mathfrak{imc}$.
Em $\mathfrak{imc}_3$ acontece a marcação dos símbolos dentro dos colchetes, tais símbolos serão marcados com a, em seguida, passa-se para $\mathfrak{imc}_4$ que é onde será escolhido qual transformação fazer com os símbolos marcados com a: $\mathfrak{dc}$ para a primeira transformação direta, $\mathfrak{rc}$ para a segunda transformação direta ou $\mathfrak{er}$ para a terceira transformação direta.
Após a transformação realizada, na função $\mathfrak{imc}_5$ os símbolos resultantes são copiados para o final da sequência e marcados com β e as modificações feitas na sequência de entrada voltam com o marcador α e os marcadores temporários são removidos.

$$
\begin{array}{rll}
\mathfrak{conv}(\mathfrak{A},\alpha,\beta,\theta) & & \mathfrak{pe}(\mathfrak{crm}(\mathfrak{conv}_1,\alpha,d),.) \\
\mathfrak{conv}_1 & & \mathfrak{ch}(\mathfrak{imc}(\mathfrak{conv}_2,\mathfrak{au},d,f,\theta),\mathfrak{re}(\mathfrak{A},d,\beta),\mathfrak{au},\theta) \\
\mathfrak{conv}_2 & & \mathfrak{e}(\mathfrak{re}(\mathfrak{conv}_1,f,d),d) \\
\mathfrak{au} & & \mathfrak{q}(\mathfrak{au}_1,.) \\
\mathfrak{au}_1 \begin{cases} & \mathfrak{B} & \mathfrak{crm}(\mathfrak{A},\alpha,\beta) \\ & \text{not } \mathfrak{B} \quad \text{R, E, R} & \mathfrak{au}_1 \end{cases} &&
\end{array}
$$

> $\mathfrak{conv}(\mathfrak{A},\alpha,\beta,\theta)$. A conversion is chosen and performed on $S(\alpha)$. The result is marked with β. $\rightarrow \mathfrak{A}$. The sequence determining the choices is $S(\theta)$. If it should happen that this sequence is exhausted before the conversion is completed then the final formula is the same as the original, i.e. $S(\alpha)$. The half finished conversion work is effectively removed from the tape by erasing the marks. [39]

A função $\mathfrak{conv}$ realiza uma conversão numa função $\lambda - K$ marcada com símbolos α, ou seja, diversas transformações imediatas são realizadas em $S(\alpha)$ até chegar em sua forma normal, onde não há mais transformações a se fazer. As operações nesta função são realizadas no fim da sequência, após o símbolo de ponto final que é escrito no fim logo no início de $\mathfrak{conv}$, e o resultado

[39] Tradução do autor: $\mathfrak{conv}(\mathfrak{A},\alpha,\beta,\theta)$. Uma conversão é escolhida e realizada em $S(\alpha)$. O resultado é marcado com β. $\rightarrow \mathfrak{A}$. A sequência determinando as escolhas é $S(\theta)$. Se acontecer desta sequência exaurir antes da conversão estar completa, então a fórmula final é a mesma que a original, i.e. $S(\alpha)$. Todo o trabalho realizado para a conversão terminada pela metade é efetivamente removida da fita apagando os marcadores.

final é marcado com β. Como também usado em algumas funções anterioes, a função $\mathfrak{conv}$ usa uma sequência de símbolos binários marcados com θ para escolher qual operação realizar e a mesma sequência também é usada para as diversas transformações imediatas que podem acontecer, caso chegue o momento que a lista de símbolos marcados com θ se esvazie (todos os símbolos são marcados com b, ver função $\mathfrak{ch}$ na seção 3.3) então a conversão é interrompida, indo para a função $\mathfrak{au}$, onde os marcadores dos símbolos escritos são apagados e, por fim, o resultado será o mesmo que a entrada, que será copiado para fim da sequência e marcado com β.

3.5 COMPUTABILIDADE DAS FUNÇÕES $\lambda - K$ DEFINÍVEIS

Nas duas seções anteriores foram definidas as funções para operar uma Máquina de Turing e, com estas, construir tabelas referentes às três regras de conversão do Cálculo $\lambda - K$, assim, contruindo uma tabela com uma função que converte uma dada função $\lambda - K$ à sua forma normal. Nesta seção é que será mostrada a computabilidade das funções $\lambda - K$, ou seja, a Máquina de Turing será capaz de computar Cálculo $\lambda - K$.
A definição de computabilidade é reiterada novamente por Turing:

> It is now comparatively simple to show that a $\lambda - K$-definable function is computable, i.e., that [...] if $f(n)$ is $\lambda - K$-definable then the sequence γ_f in which there are $f(n)$ figures 1 between the nth and the $(n+1)$th 0, and $f(0)$ figures before the first 0, is computable. [40]

Logo, dada uma função qualquer que é $\lambda - K$-definível, ela consegue escrever todos os resultados possíveis, cujas entradas são números naturais, cada resultado separado por 1, em outras palavras, existe um algoritmo capaz retornar a saída de acordo com a entrada especificada. Esta definição de computabilidade foi usada por Turing em *Computable* para a prova da resposta negativa ao problema de decisão proposto por Hilbert.
Um exemplo para ilustrar a definição, seja a seguinte função $f(n)$:

$$f(n) = 2n + 1.$$

Uma função matemática relaciona um elemento de um conjunto de entrada (o domínio) com um elemento de outro conjunto que será a saída (o contradomínio), portanto, aplicando um n de entrada resultará no número corres-

[40] Tradução do autor: Agora é comparativamente simples de mostrar que uma função $\lambda - K$-definível é computável, i.e., que [...] se $f(n)$ é $\lambda - K$-definível então a sequência γ_f em que $f(n)$ figura 1 entre o n-ésimo e o $(n+1)$-ésimo 0 e $f(0)$ figura antes do primeiro 0, é computável.

pondente ao contradomínio definido na lei de formação "$2n+1$". Aplicando alguns examplos de n na função $f(n)$ teremos:

$$f(0) = 2.0+1 = 1$$
$$f(1) = 2.1+1 = 3$$
$$f(2) = 2.2+1 = 5$$
$$f(3) = 2.3+1 = 7.$$

Assim, a máquina que é capaz de computar a função $f(n)$ deve escrever cada um dos resultados acima com a quantidade de zeros necessária, cada resultado separado por 1. Portanto, a máquina deve escrever

$$0100010000010000000,$$

que seria a sequência γ_f.
A seguir, mais duas tabelas são definidas antes da função que computa Cálculo $\lambda - K$.

> To simplify the table for the machine which computes γ_f we use the abbreviation $\mathfrak{Wr}(\mathfrak{A}, M, \alpha)$ for an internal configuração starting from which the machine writes the sequence M of symbols at the end, making it with α and finishing in the internal configuration $\mathfrak{A}$. Thus the table for $\mathfrak{Wr}(\mathfrak{A}, \lambda x', \alpha)$ would be: [41]
>
> | | | |
> |---|---|---|
> | $\mathfrak{Wr}(\mathfrak{A}, \lambda x', \alpha)$ | | $\mathfrak{pe}(\mathfrak{Wr}_1, \mathfrak{B})$ |
> | $\mathfrak{Wr}_1$ | Pλ, R, Pα, R, Px, R, Pα, R, P$'$, R, Pα | $\mathfrak{A}$ |
>
> We use one more skeleton table: [42]
>
> $\mathfrak{pls}(\mathfrak{A}, \alpha, \beta)$ $\quad$ $\mathfrak{funk}(\mathfrak{e}(\mathfrak{re}(\mathfrak{A}, a, \alpha), \alpha), \beta, \alpha, a)$

A primeira função, $\mathfrak{Wr}$, simplesmente imprime a função $\lambda - K\,M$ definida e a marca com α, na definição, a função $\lambda - K\,M$ é "$\lambda x'$". A segunda função, $\mathfrak{pls}$, coloca duas sequências marcadas com α e β, respectivamente, entre colchetes e marca-os com a, apaga os marcadores α que tiverem na fita e, por fim, remarca a mesma sequência $[S(\beta)][S(\alpha)]$ de a para α.

> If F is the formula which $\lambda - K$-defines $f(n)$ then the table for the machine which computes γ_f is: [43]

[41] Tradução do autor: Para simplificar a tabela para a máquina que computa γ_f nós usamos a abreviação $\mathfrak{Wr}(\mathfrak{A}, M, \alpha)$ para uma configuração iniciando com a máquina escrevendo a sequência M de símbolos no fim, fazendo isso com α e terminando na configuração interna $\mathfrak{A}$. Portanto a tabela para $\mathfrak{Wr}(\mathfrak{A}, \lambda x', \alpha)$ será:

[42] Tradução do autor: Nós usaremos mais uma *skeleton table*:

[43] Tradução do autor: Se F é a fórmula que $\lambda - K$-define $f(n)$ então a tabela para a máquina que computa γ_f é:

$\mathfrak{b}$		Pə, R, Pə	$\mathfrak{Wr}(\mathfrak{b}_1, F, h)$
$\mathfrak{b}_1$			$\mathfrak{Wr}(\mathfrak{b}_2, \lambda x[\lambda x'[x']], i)$
$\mathfrak{b}_2$			$\mathfrak{crm}(\mathfrak{b}_3, i, k)$
$\mathfrak{b}_3$			$\mathfrak{Wr}(\mathfrak{ba}, \lambda x''[\lambda x[\lambda x'[[x][[[x'']][x]][x']]]], u)$
$\mathfrak{ba}$			$\mathfrak{funk}(\mathfrak{cn}_1, h, k, v)$
$\mathfrak{cn}$			$\mathfrak{add}(\mathfrak{cn}_1, s, L, M)$
$\mathfrak{cn}_1$			$\mathfrak{crm}(cn_2, i, d)$
$\mathfrak{cn}_2$			$\mathfrak{ch}(\mathfrak{re}(\mathfrak{cn}_3, d, m), \mathfrak{pls}(\mathfrak{cn}_2, d, u), \mathfrak{cn}_6, s)$
$\mathfrak{cn}_3$			$\mathfrak{conv}(\mathfrak{cn}_4, v, w, s)$
$\mathfrak{cn}_4$			$\mathfrak{cpr}(\mathfrak{cn}_5, \mathfrak{cn}_6, w, m)$
$\mathfrak{cn}_6$			$\mathfrak{e}(\mathfrak{e}(\mathfrak{e}(\mathfrak{cn}_{10}, w), m), d)$
$\mathfrak{cn}_{10}$			$\mathfrak{ch}(\mathfrak{cn}_{10}, \mathfrak{cn}_{10}, \mathfrak{cn}, s)$
$\mathfrak{cn}_5$			$\mathfrak{q}(\mathfrak{cn}_7, m)$
$\mathfrak{cn}_7$		E	$\mathfrak{q}(\mathfrak{cn}_8, m)$
$\mathfrak{cn}_8$		E	$\mathfrak{q}(\mathfrak{l}(\mathfrak{cn}_9), m)$
$\mathfrak{cn}_9$	]	R, E	$\mathfrak{pem}(\mathfrak{q}(\mathfrak{l}(\mathfrak{cn}_9), m), 1, a)$
	not]		$\mathfrak{pem}(\mathfrak{e}(\mathfrak{e}(\mathfrak{e}(\mathfrak{e}(\mathfrak{ba}_1, s), v), w), m), 0, a)$
$\mathfrak{ba}_1$			$\mathfrak{pls}(\mathfrak{ba}, k, u)$

When the machine reaches the internal configuration $\mathfrak{ba}$ for the $(n+1)$th time $(n \geq 0)$ the tape bears the formula F marked with h, the formula n representing the natural number n (or rather a formula convertible into it) marked with k, 0 marked with i, and S marked with u. [44]

A tabela escreve, portanto, a sequência γ_f que separados por zeros representa cada um dos resultados possíveis de cada entrada natural de uma função $\lambda - K$ qualquer, tal função é representado por F na tabela. Outra observação é o uso de uma função não definida por Turing, porém, tem propósito intuitivo: a função $\mathfrak{cpr}(\mathfrak{A}, \mathfrak{B}, \alpha, \beta)$ simplesmente compara a sequência $S(\alpha)$ com a sequência $S(\beta)$, se ambas forem iguais, então segue para $\mathfrak{A}$, caso contrário, segue para $\mathfrak{B}$. Durante a execução das funções especificadas na tabela, a máquina novamente vai lidar com uma lista de símbolos binários para decidir qual caminho tomar, cada elemento da lista com símbolos binários L e M é marcado com s.

Nota-se que as funções iniciais da tabela, de $\mathfrak{b}$ até $\mathfrak{b}_3$ são funções de preparação da máquina, assim, elas só são usadas na inicialização.

Iniciando a computação em $\mathfrak{b}$, dois xevás são escritos na fita e segue a execução para $\mathfrak{Wr}$. O padrão de escrever dois xevás no início da fita já foi usado em

[44] Tradução do autor: Quando a máquina atinge a configuração interna $\mathfrak{ba}$ pela $(n+1)$-ésima vez $(n \geq 0)$ a fita carrega a fórmula F marcada com h, a fórmula n representando o número natural n (ou em vez uma fórmula convertível para isto) marcada com k, 0 marcado com i e S marcado com u.

Computable, para ter clara indicação onde é o começo da sequência. Vamos supor que a fórmula F define a seguinte função $\lambda - K$: $\lambda x[x]$, assim, após $\mathfrak{Wr}$ o conteúdo escrito na fita, por enquanto, está ilustrado na figura 14.

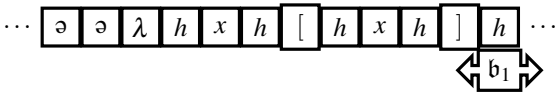


Figura 14: Conteúdo da fita com a máquina na configuração $\mathfrak{b}_1$.

Em $\mathfrak{b}_1$ a função $\lambda - K$ $\lambda x[\lambda x'[x']]$ é marcada com i, esta função representa uma abreviação do número zero, como apresentado na seção 3.2 sobre as restrições do cálculo $\lambda - K$. Em $\mathfrak{b}_2$ copia-se a função $\lambda - K$ que representa o número zero e a marca com k, será com esta nova cópia com que a função irá trabalhar e manipular, ou seja, será o n de entrada da função $f(n)$ - após cada conversão de $f(n)$ ser realizada, n será incrementado em mais um. Em $\mathfrak{b}_3$ escreve a função e a marca com u, esta função respresenta a função *sucessora*, ou seja, incrementa mais um, como mostrado numa aplicação exemplo na seção 3.2. Uma observação é que função sucessora também é chamada de função S, que é diferente de $S(\alpha)$, que representa uma sequência de símbolos marcados com α.
Na configuração $\mathfrak{ba}$, a função $\mathfrak{funk}$ colocará as sequências $S(h)$ e $S(k)$ entre colchetes e marcará isso tudo com v, ou seja, está montando a função $[F][0]$ que, no exemplo usado aqui, representa $[\lambda x[x]][\lambda x[\lambda x'[x']]]$.
Em $\mathfrak{cn}_1$ simplesmente é realizada uma cópia dos símbolos da sequência $S(i)$ para o fim da sequência e são marcados com d. Em $\mathfrak{cn}_2$ será realizada a escolha entre seguir para uma de três configurações possíveis: L para seguir adiante com a conversão atual da função marcada com v, substituindo os símbolos marcados com d para m e indo para a configuração $\mathfrak{cn}_3$; M para uma validação, mais adiante haverá uma comparação de resultados para garantir que a conversão realizada está correta e a função em $\mathfrak{pls}(\mathfrak{cn}_2, d, u)$ será repetida até chegar no mesmo valor de entrada marcado com k, com o resultado marcado com d; caso a lista s esteja toda preenchida, a computação até então é interrompida, com os marcadores d, m e w sendo apagados e a lista s atualizada.

> This brings us to the internal configuration $\mathfrak{cn}_3$. A conversion is then chosen and performed on $S(v)$, i.e. on $[F][n]$. The result is marked with w and compared with $S(m)$. If they are not alike the letters w, m are erased and we go back to $\mathfrak{cn}_1$ after transforming the sequence $S(s)$ which determines the choices into the

> next sequence. [45]

Em $\mathfrak{cn}_3$ é realizada a conversão de $[F][0]$, resultando em 0, assim o resultado "$\lambda x[\lambda x'[x']]$" será escrito na fita e cada símbolo é marcado com w. Em $\mathfrak{cn}_4$ é feita a validação, o resultado da conversão no passo anterior ($S(w)$) com a sequência resultante em $\mathfrak{cn}_2$ ($S(m)$): caso as duas sequências sejam diferentes, então passa-se para a configuração $\mathfrak{cn}_6$ e segue-se similar ao caso onde a lista esteja vazia em $\mathfrak{cn}_2$; caso as duas sequências sejam iguais, então segue para $\mathfrak{cn}_5$.

> If they are alike then 1 is written at the end repeated r times followed by 0, all of which is marked with a. In order to have the correct number of figures we make use of the fact that the number of brackets occurring consecutively at the end of $S(m)$ is $r+2$. The machine is back in the internal configuration $\mathfrak{ba}$ as soon as $S(k)$ has been changed to $[S][S(k)]$. [46]

Em $\mathfrak{cn}_9$ que é realizado a escrita da sequência γ_f, onde cada símbolo de fecha colchetes restante representa uma unidade do resultado que será escrito como 1 na fita - em $\mathfrak{cn}_7$ e $\mathfrak{cn}_8$ dois símbolos de fecha colchetes mais a direita são apagados. O resultado $\lambda x[\lambda x'[x']]$ sem os dois símbolos mais a direita representa zero, logo, nenhum 1 é escrito, um 0 é escrito na fita, que será o separador de cada resultado e os marcadores m, w, v e s são removidos e segue-se para $\mathfrak{ba}_1$. Em $\mathfrak{ba}_1$ é realizado o sucessor de n, ou seja, $S(k)$ passa a ser $[S][0]$, logo, 1 ou $\lambda x[\lambda x'[[x][x']]]$, seguindo para $\mathfrak{ba}$ e todo o processo é repetido. Quando chegar em $\mathfrak{cn}_{10}$ novamente, removendo os dois "]" mais a direita de 1, resultará em apenas um colchete, logo, o número 1. A sequência γ_f como escrita na fita é ilustrada na figura 15.

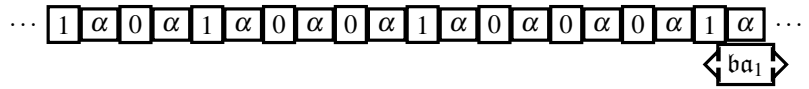


Figura 15: Trecho da sequência γ_f na fita.

[45] Tradução do autor: Isso nos traz à configuração interna $\mathfrak{cn}_3$. Uma conversão é então escolhida e realizada em $S(v)$, i.e. em $[F][n]$. O resultado é marcado com w e comparado com $S(m)$. Se não são iguais, as letras w, m são apagados e voltaremos a $\mathfrak{cn}_1$ depois de transformar a sequência $S(s)$ que determina as escolhas para a próxima sequência.

[46] Tradução do autor: Se são iguais, então 1 é escrito no fim e repetido r vezes seguido de 0, todos são marcados com a. A fim de manter o número correto de figuras, faremos uso do fato que o número de colchetes ocorrendo consecutivamente no fim de $S(m)$ é $r+2$. A máquina volta para a configuração interna $\mathfrak{ba}$ assim que $S(k)$ foi substituído por $[S][S(k)]$.

> No attempt is being made to give a formal proof that this machine has the properties claimed for it. Such a formal proof is not possible unless the ideas of substitution and so forth occuring in the definition of conversion are formally defined and the best form of such a definition is possibly in terms of machines.
>
> If $f(n)(n \geq 1)$ is λ-definable, i.e. if F is well-formed [...], then the present argument shows also that $f(n)$ is then computable in sense that a function $g(n)$ of positive integers is computable if there is a computable sequence with $g(n)$ figures 1 between the nth and $(n+1)$th figure 0. [47]

Com a tabela especificando a computabilidade das funções $\lambda - K$, nenhuma outra definição formal é necessária, pois as tabelas especificadas em seções anteriores a complementa. As definições primárias de cálculo $\lambda - K$ que são os conceitos de fórmulas bem-formadas e as regras de convertibilidade, que envolvem questões como substituição e transformação, já foram definidos antes - reforçando o caso da implementação das conversões na seção 3.4 e as restrições de uma implementação de uma fórmula bem-formada em uma máquina na seção 3.2 - que tudo isso por si só já provam que a máquina é capaz de computar funções $\lambda - K$.
Desse modo, temos uma máquina de computar cálculo $\lambda - K$, portanto, qualquer problema $\lambda - K$-definível (que é equivalente à λ-definível) a máquina é capaz de computar; por outro lado, tendo um modelo $\lambda - K$-definível que é utilizado por uma máquina, podemos adaptar problemas computáveis para a máquina que computa cálculo $\lambda - K$, assim, um problema computável pode ser transformado em um problema λ-definível, complementando a equivalência.

[47] Tradução do autor: Nenhuma tentativa foi feita de dar uma prova formal de que esta máquina tem as propriedades clamadas para si. Tal prova formal não é possível a menos que as ideias de substituição e assim por diante ocorrendo na definição de conversão são formalmente definidos e a melhor forma de tal definição é possivelmente em termos de máquinas.

Se $f(n)(n \geq 1)$ é λ-definível, i.e. se F é bem formado [...] então o presente argumento também mostra que $f(n)$ é então computável no sentido que a função $g(n)$ dos inteiros positivos é computável se tem uma sequência computável com $g(n)$ figurando 1 entre o n-ésimo e o $(n+1)$-ésimo 0.

4 CONCLUSÃO

O décimo problema de Hilbert apresentava um problema envolvendo usar uma série de passos para determinar se uma dada equação diofantina é resolvível em inteiros racionais, assim, a resposta para este problema seria capaz de *decidir* se dada equação é ou não resolvível, logo, a resposta faria parte de um subconjunto de operações efetivamente calculáveis - seria capaz de processar o problema e decidir uma resposta sobre ele. Em 1934, Kurt Gödel, com base no trabalho de Jacques Herbrand, formulou as funções recursivas, também chamadas de recursividade de Gödel-Herbrand. Com o intuito de também definir equações efetivamente calculáveis, Alonzo Church formulou o Cálculo λ, assim, vindo a apresentar o conceito de λ-definível. Stephen Kleene e Church, independentes, viriam provar a equivalência entre λ-definível e a recursividade de Gödel-Herbrand.
Outro problema de Hilbert, chamado de *Entscheidungsproblem*, perguntava da decibilidade da desmontração de uma fórmula em lógica de primeira ordem, em 1936, Church usou modelos λ-definíveis para responder negativamente à pergunta. Independente a Church e pouco depois da resposta deste, Alan Turing formulou o modelo de máquinas de computar que também resulta na resposta negativa ao *Entscheidungsproblem*. Turing adicionou um apêndice em sua publicação referente ao *Entscheidungsproblem* sobre a equivalência entre os modelos computáveis e λ-definíveis e, depois, detalhou essa prova que é o objeto de estudo deste trabalho. A última seção da publicação de Turing apresenta sobre a recursidade das funções computáveis, complementando a prova mostrando a equivalência entre funções computáveis e recursivas. Esta seção foi omitida neste trabalho por questões de escopo, o objetivo deste trabalho foi apresentar entre os modelos de Máquinas de Turing e Cálculo λ, no entanto, está aberta a possibilidade de trabalho futuro investigar as funções recursivas, além de estudar a publicação de Kleene (KLEENE, 1936) sobre a equivalência de funções λ-definíveis e recursivas.
Para provar a equivalência entre os modelos computáveis e λ-definíveis, Turing construiu uma máquina que é capaz de computar Cálculo λ. Primeiro ele definiu um modelo de Cálculo λ lexicalmente mais restrito, é equivalente ao modelo de Church, porém, usa menos símbolos para representar as funções, esse modelo de Turing é chamado de Cálculo $\lambda - K$. Em seguida, usando funções que já tinham sido definidas no artigo anterior, *Computable* (TURING, 1937b), Turing construiu novas funções e complementou as existentes para lidar com sequências, além de símbolos individuais. Com as funções e o conceito de Cálculo $\lambda - K$ bem definidos, mais três tabelas referentes às três regras de conversão de funções $\lambda - K$-definíveis foram construídas para

que operações possam ser feitas. Por fim, com as funções criadas e o modelo $\lambda - K$ bem definido, restou apenas a construção da função final que computa funções $\lambda - K$ quaisquer.

O Cálculo λ proposto por Church em 1936, também chamado de Cálculo λ tipado, mostra que uma variável x pode ser aplicada apenas nas variáveis livres x na função $\lambda x.M$. Com influências de Haskell Curry, foi proposto o Cálculo $\lambda - K$ (diferente do modelo de mesmo nome criado por Turing e estudado neste trabalho) onde x pode ser aplicado nas variáveis livres da função $\lambda x.M$ mesmo que não existam ocorrências de x em M, um método não tipado. Existem diversos modelos de cálculo λ tipados e não tipados, tal paradigma servindo de inspiração para as linguagens de programação tipadas e não tipadas (BARENDREGT, 1997), no entanto, o Cálculo λ definiu principalmente o paradigma funcional das linguagens de programação.

Assim, em 1936 existiam três modelos teóricos sobre a questão de equações efetivamente calculáveis: as Máquinas de Turing, as funções recursivas e as funções λ-definíveis ou Cálculo λ. Em 1952, no livro *Introduction to Metamathematics*, Kleene reformula o modelo de uma máquina de Turing, a máquina contém um cabeçote que move para a esquerda e para a direita e lê e escreve símbolos na fita, porém contém apenas um símbolo, que é um risco vertical, onde um risco escrito na fita representa o número zero, dois riscos escritos representam o número um e assim sucessivamente. A máquina reformulada também não contém uma configuração ou estado específico para parar a máquina, como as letras alemãs definiam no trabalho de Turing, se a máquina fosse para uma próxima configuração inexistente, então ela pararia ou retornaria o resultado. As entradas especificadas na máquina já estariam codificadas nela mesma, ao contrário de uma fita vazia. Por fim, para Kleene uma máquina adequada é uma que resolve o problema e para, máquinas que ficam executando infinitamente não são adequadas, portanto, as máquinas de Kleene já são um pouco mais fiéis ao comportamento esperado dos computadores de hoje (PETZOLD, 2008).

Martin Davis já considerava sobre como determinar quando uma máquina de Turing irá terminar a execução e, no seu livro *Computability and Unsolvability* (1958), cunhou o termo "problema da parada" - o modelo de máquina de Turing usado no livro é similar ao modelo reformulado por Kleene. Segundo (PETZOLD, 2008), *Computability and Unsolvability* pode ser considerado como o livro que iniciou o estudo de computabilidade como tópico.

Em 1979, num artigo de autoria de Robert W. Floyd (1936-2001) é discutido sobre o tópico de paradigmas na programação (FLOYD, 1979). Na publicação, Floyd analisa sobre as técnicas implementadas por programadores e a forma que os professores ensinam programação, disso discute os padrões dos programas resultantes dessas duas abordagens, fazendo um paralelo com

o livro *The Structure of Scientific Revolutions* de Thomas S. Kuhn. No fim, Floyd faz um apelo para que cada um, seja professor, programador ou designer de linguagens de programação, que especifiquem e tentem notar padrões que as linguagens possuem e como isso pode ser vantajoso, ou seja, deixar claro os paradigmas que estão sendo usados e em qual momento eles são proveitosos. O Cálculo λ influenciou o desenvolvimento das linguagens de programação, é notável a influência direta em linguagens que usam o paradigma funcional, tais como Haskell, F# e Erlang. Indiretamente, também influenciou as linguagens de paradigma imperativo, como apresentado por Peter Landin (1930-2009) em (LANDIN, 1965a) e (LANDIN, 1965b), mostrando uma correspondência entre expressões na linguagem ALGOL 60 e uma versão modificada do Cálculo λ, assim, ele notou que o modelo formal de Church pode ser usado para modelar uma linguagem de programação. O ALGOL 60 influenciou muitas linguagens imperativas posteriores, tais quais Pascal, Ada e C.

Como trabalhos futuros a este, além do estudo de funções recursivas de Gödel e Herbrand antes mencionado, outro tópico de estudo são os problemas de Hilbert propostos em 1900. Em (YANDELL, 2002, p.385) há uma discussão sobre os vinte e três problemas, o autor relata que dezesseis deles foram resolvidos de forma discreta, ou seja, respondem ao problema proposto por Hilbert e dificilmente as respostas terão alguma outra grande modificação no futuro - são os problemas 1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 15, 17, 18, 21 e 22. No entanto, os problemas 1, 2, 5, 9, 15, 18 e 22 tiveram respostas parciais, por exemplo, tiveram mais de uma resposta aceitável ou dependendo da interpretação do problema a resposta resolve o problema, caso contrário não resolve; quatro problemas são vagos em suas proposições, apesar de terem tido avanços nas respostas, o que abre a dúvida sobre a resolubilidade dos problemas - os problemas 12, 19, 20 e 23; por fim, três problemas não foram resolvidos - os problemas 6, 8 e 16. Em (KINYON; BRUMMELEN, 2005, p.266) resume, "grosseiramente" segundo os editores, as áreas em que os problemas de Hilbert estão abordando: os problemas de fundamentos da matemática são 1, 2, 6 e o cancelado problema 24; os problemas de análise matemática são 19, 20, 21, 22 e 23; os problemas de geometria são 3, 4, 15 e 18; e os problemas de aritmética e álgebra são 5, 7, 8, 9, 10, 11, 12, 13, 14, 16 e 17.

REFERÊNCIAS

BARENDREGT, H. The impact of lambda calculus in logic and computer science. **The Bulletin of Symbolic Logic**, v. 3, n. 2, p. 181–215, jun. 1997. ISSN 10798986. Disponível em: <http://www.jstor.org/stable/421013>.

BARENDREGT, H. P. **The Lambda Calculus**: Its syntax and semantics. [S.l.]: Elsevier, 1984. (Studies in Logic and The Foundations of Mathematics, v. 103). ISBN 0444867481.

BARENDREGT, H. P. Functional programming and lambda calculus. In: LEEUWEN, J. V. (Ed.). **Handbook of Theoretical Computer Science**. 1ª. ed. Cambridge, MA, EUA: The MIT Press, 1994. B, cap. 7, p. 320–364. ISBN 0444880747.

CASTI, J. L. **The Five Golden Rules**: Great theories of 20th-century mathematics - and why they matter. [S.l.]: John Wiley & Sons, Inc., 1996. ISBN 0471002615.

CHURCH, A. A set of postulates for the foundation of logic. **Annals of Mathematics**, v. 33, n. 2, p. 346–366, abr. 1932.

CHURCH, A. An unsolvable problem of elementary number theory. **American Journal of Mathematics**, Johns Hopkins University Press, v. 58, n. 2, p. 345–363, 1936. ISSN 00029327, 10806377. Disponível em: <http://www.jstor.org/stable/2371045>.

CHURCH, A.; ROSSER, J. B. Some properties of conversion. **Transactions of the American Mathematical Society**, American Mathematical Society, v. 39, n. 3, p. 472–482, 1936. ISSN 00029947. Disponível em: <http://www.jstor.org/stable/1989762>.

COSTA, N. C. A. da. **Introdução aos Fundamentos da Matemática**. 4ª. ed. São Paulo, SP, Brasil: Editora Hucitec, 2008. ISBN 8527101831.

DAVIS, M. **The Undecidable**: Basic papers on undecidable propositions, unsolvable problems and computable functions. [S.l.]: Raven Press Books, Ltd., 1965.

FILHO, C. F. **História da Computação**: O caminho do pensamento e da tecnologia. Porto Alegre, RS, Brasil: EDIPUCRS, 2007. ISBN 9788574306919.

FLOYD, R. W. The paradigms of programming. **Communications of the ACM**, ACM, Nova Iorque, NY, EUA, v. 22, n. 8, p. 455–460, ago. 1979. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359138.359140>.

HEIJENOORT, J. van. **From Frege to Gödel**: A source book in mathematical logic, 1879-1931. [S.l.]: Harvard University Press, 1967. 83–97 p.

HILBERT, D. Lecture delivered before the international congress of mathematicians at paris in 1900. **Bulletin of the American Mathematical Society**, v. 8, n. 10, 1902. Disponível em: <http://www.ams.org/journals/bull/1902-08-10/S0002-9904-1902-00923-3-/S0002-9904-1902-00923-3.pdf>.

HILBERT, D.; ACKERMANN, W. **The Principles of Mathematical Logic**. Nova Iorque, NY, EUA: Chelsea Publishing Company, 1950.

HINDLEY, J. R.; SELDIN, J. P. **Lambda-Calculus and Combinators**: An introduction. 1ª. ed. Nova Iorque, NY, EUA: Cambridge University Press, 2008. ISBN 9780521898850.

HODGES, A. **Alan Turing: The Enigma**. [S.l.]: Princeton University Press, 1983. ISBN 9780691164724.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to Automata Theory, Languages, and Computation**. Boston, MA, EUA: Addison-Wesley, 2001. ISBN 0201441241.

IRVINE, A. D.; DEUTSCH, H. Russell's paradox. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Winter 2016. [S.l.]: Metaphysics Research Lab, Stanford University, 2016.

KINYON, M.; BRUMMELEN, G. van (Ed.). **Mathematics and the Historian's Craft**: The kenneth o. may lectures. 1ª. ed. Nova Iorque, NY, EUA: Springer-Verlag New York, 2005. (CMS Books in Mathematics). ISSN 1613-5237. ISBN 978-0-387-28272-5.

KLEENE, S. C. A theory of positive integers in formal logic. part i. **American Journal of Mathematics**, The Johns Hopkins University Press, v. 57, n. 1, p. 153–173, jan. 1935. Disponível em: <https://www.jstor.org/stable/2372027>.

KLEENE, S. C. λ -definability and recursiveness. **Duke Mathematical Journal**, Duke University Press, v. 2, n. 2, p. 340–353, jun. 1936. Disponível em: <https://doi.org/10.1215/S0012-7094-36-00227-2>.

KLEENE, S. C. Recursive predicates and quantifiers. **Transactions of the American Mathematical Society**, American Mathematical Society, v. 53, n. 1, p. 41–73, 1943. Disponível em: <http://www.jstor.org/stable/1990131>.

KNUTH, D. E. **The Art of Computer Programming**. 3ª. ed. [S.l.]: Addison Wesley Longman, 1997. ISBN 0201896834.

KOZEN, D. C. **Automata and Computability**. [S.l.]: Springer, 1997. ISBN 0387949070.

LANDIN, P. J. Correspondence between algol 60 and church's lambda-notation: Part i. **Communications of the ACM**, ACM, Nova Iorque, NY, EUA, v. 8, n. 2, p. 89–101, fev. 1965. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/363744.363749>.

LANDIN, P. J. A correspondence between algol 60 and church's lambda-notations: Part ii. **Communications of the ACM**, ACM, Nova Iorque, NY, EUA, v. 8, n. 3, p. 158–167, mar. 1965. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/363791.363804>.

LINK, G. **One Hundred Years of Russell's Paradox**. Berlim, Alemanha: Walter de Gruyter GmbH, 2004. ISBN 3110174383.

O'DONNEL, M. J. **Computing in systems described by equations**. Heidelberg, Baden-Württemberg, Alemanha: Springer-Verlag, 1977. (Lecture Notes in Computer Science). ISBN 3540085319.

PETZOLD, C. **The Annotated Turing**: A guided tour through alan turing's historic paper on computability and the turing machine. Indianópolis, IN, EUA: Wiley Publishing, Inc, 2008. ISBN 9780470229057.

PIERCE, B. C. **Types and Programming Languages**. Cambridge, MA, EUA: The MIT Press, 2002. ISBN 0262162091.

RUSSELL, B. **The Principles of Mathematics**. Nova Iorque, NY, EUA: W. W. Norton & Company, 1903. ISBN 0393314049.

RUSSELL, B. **The Problems of Philosophy**. [S.l.: s.n.], 1912.

SILVA, T. Oliveira e; HERZOG, S.; PARDI, S. Empirical verification of the even Goldbach conjecture and computation of prime gaps up to $4 \cdot 10^{18}$. **Mathematics of Computation**, v. 83, n. 288, p. 2033–2060, jul. 2014.

SINGH, S. **O Último Teorema de Fermat**. 1ª. ed. Rio de Janeiro, RJ, Brasil: Editora Record, 2014. ISBN 9788577994625.

SIPSER, M. **Introduction to the Theory of Computation**. 3ª. ed. [S.l.]: Cengage Learning, 2013. ISBN 978113318790.

STEWART, J. **Calculus**. 6ª. ed. Belmont, CA, EUA: Thomson, 2008. ISBN 0495011606.

THIELE, R. Hilbert's twenty-fourth problem. **The American Mathematical Monthly**, Mathematical Association of America, p. 1–24, jan. 2003. Disponível em: <https://www.maa.org/programs/maa-awards-/writing-awards/hilberts-twenty-fourth-problem>.

TURING, A. M. Computability and λ-definability. **The Journal of Symbolic Logic**, v. 2, p. 153–163, dez. 1937.

TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. **Proceedings of the London Mathematical Society**, s2-42, n. 1, p. 230–265, 1937. Disponível em: <https://londmathsoc-.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-42.1.230>.

YANDELL, B. H. **The Honor Class**: Hilbert's problems and their solvers. Natick, MA, USA: A K Peters, Ltd., 2002. ISBN 1568812167.

APÊNDICE A – TABELAS

Este apêndice apresenta as tabelas contendo as definições das funções descritas por Turing na publicação *Computable*(TURING, 1937b) que são usadas na investigação da equivalência no capítulo 3.

A.1 *PRINT AT THE END*

$$\mathfrak{pe}(\mathfrak{C},\beta) \qquad\qquad \mathfrak{f}(\mathfrak{pe}_1(\mathfrak{C},\beta),\mathfrak{C},\mathfrak{d})$$

$$\mathfrak{pe}_1(\mathfrak{C},\beta)\begin{cases} \text{Any} & \text{R, R} & \mathfrak{pe}_1(\mathfrak{C},\beta) \\ \text{None} & \text{P}\beta & \mathfrak{C} \end{cases}$$

From $\mathfrak{pe}(\mathfrak{C},\beta)$ the machine prints β at the end of the sequence of symbols and $\rightarrow \mathfrak{C}$. [1]

A função $\mathfrak{pe}$ imprime o marcador especificado no segundo parâmetro, neste caso β, no final de uma sequência de símbolos. Primeiramente, é procurado o símbolo xevá ($\mathfrak{d}$), que indica o início da fita, ao ser encontrado, o cabeçote da máquina é movimentado para a direita toda a vez que é encontrado algum símbolo. Se nenhum símbolo estiver escrito na fita, a máquina escreve β no espaço e passa para a configuração $\mathfrak{C}$, terminando a computação.

$\mathfrak{pe}_2(\mathfrak{C},\alpha,\beta) \quad \mathfrak{pe}(\mathfrak{pe}(\mathfrak{C},\beta),\alpha)$

$\mathfrak{pe}_2(\mathfrak{C},\alpha,\beta)$. The machine prints $\alpha\ \beta$ at the end. [2]

A função $\mathfrak{pe}_2$ é uma extensão da função $\mathfrak{pe}$, no final são dois símbolos que são escritos no fim da sequência, sem espaço vazio entre α e β.

A.2 *LEFT AND RIGHT*

$\mathfrak{l}(\mathfrak{C})$	L	$\mathfrak{C}$
$\mathfrak{r}(\mathfrak{C})$	R	$\mathfrak{C}$
$\mathfrak{f}'(\mathfrak{C},\mathfrak{B},\alpha)$		$\mathfrak{f}(\mathfrak{l}(\mathfrak{C}),\mathfrak{B},\alpha)$
$\mathfrak{f}''(\mathfrak{C},\mathfrak{B},\alpha)$		$\mathfrak{f}(\mathfrak{r}(\mathfrak{C}),\mathfrak{B},\alpha)$

From $\mathfrak{f}'(\mathfrak{C},\mathfrak{B},\alpha)$ it dows the same as for $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$ but moves to the left before $\rightarrow \mathfrak{C}$. [3]

---

[1] Tradução do autor: De $\mathfrak{pe}(\mathfrak{C},\beta)$ a máquina imprime β no fim da sequência de símbolos e $\rightarrow \mathfrak{C}$.

[2] Tradução do autor: $\mathfrak{pe}_2(\mathfrak{C},\alpha,\beta)$. A máquina imprime $\alpha\ \beta$ no fim.

[3] Tradução do autor: De $\mathfrak{f}'(\mathfrak{C},\mathfrak{B},\alpha)$ realiza o mesmo de $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$, mas move o cabeçote para a esquerda antes $\rightarrow \mathfrak{C}$.

As funções $\mathfrak{l}$ (*left*) e $\mathfrak{r}$ (*right*) movem o cabeçote para a esquerda e para a direita, respectivamente. Ambas as funções foram incorporadas à função *find*, resultando em duas novas, $\mathfrak{f}'$ e $\mathfrak{f}''$. As funções são similares à função *find* (descrito em 3.3), mas após o símbolo desejado ser encontrado, o cabeçote da máquina é movido para a esquerda ($\mathfrak{f}'$) ou para a direita ($\mathfrak{f}''$).

A.3 *COPY*

$$\begin{array}{ccc} \mathfrak{c}(\mathfrak{C},\mathfrak{B},\alpha) & & \mathfrak{f}'(\mathfrak{c}_1(\mathfrak{C}),\mathfrak{B},\alpha) \\ \mathfrak{c}_1(\mathfrak{C}) & \beta & \mathfrak{pe}(\mathfrak{C},\beta) \end{array}$$

$\mathfrak{c}(\mathfrak{C},\mathfrak{B},\alpha)$. The machine writes at the end the first symbol marked α and $\rightarrow \mathfrak{C}$.

The last line stands for the totality of lines obtainable from it by replacing β by any symbol which may occur on the tape of the machine concerned. [4]

A função *copy* escreve o primeiro símbolo marcado com α no fim da sequência de símbolos, ou seja, copia-o para o fim da fita. Da função $\mathfrak{c}$, indo para $\mathfrak{f}'$ a máquina acha pelo primeiro símbolo marcado com α e move o cabeçote para a esquerda e passa a computação para $\mathfrak{c}_1$, onde o símbolo apontado pelo cabeçote é passado para $\mathfrak{pe}$ que o imprime no final da fita. O β que aparece na linha da função $\mathfrak{c}_1$ indica o símbolo que está sendo apontado pelo cabeçote será copiado para o fim da fita.

A.4 *COPY AND ERASE*

$$\begin{array}{ll} \mathfrak{ce}(\mathfrak{C},\mathfrak{B},\alpha) & \mathfrak{c}(\mathfrak{e}(\mathfrak{C},\mathfrak{B},\alpha),\mathfrak{B},\alpha) \\ \mathfrak{ce}(\mathfrak{B},\alpha) & \mathfrak{ce}(\mathfrak{ce}(\mathfrak{B},\alpha),\mathfrak{B},\alpha) \end{array}$$

$\mathfrak{ce}(\mathfrak{B},\alpha)$. The machine copies down in order at the end of all symbols marked α and erases the letters α; $\rightarrow \mathfrak{B}$. [5]

Na função *copy and erase* a máquina copia, da esquerda para a direita, todos os símbolos marcados com α para o fim da fita e apaga os marcadores α. Há duas versões da função $\mathfrak{ce}$, com dois e três parâmetros. A função de três

[4] Tradução do autor: $\mathfrak{c}(\mathfrak{C},\mathfrak{B},\alpha)$. A máquina escreve no fim o primeiro símbolo marcado com α e $\rightarrow \mathfrak{C}$.

A última linha significa a totalidade de linhas obtíveis disso trocando β por qualquer outro símbolo que possa ocorrer na fita da máquina em questão.

[5] Tradução do autor: $\mathfrak{ce}(\mathfrak{B},\alpha)$. A máquina copia em ordem ao fim todos os símbolos marcados com α e apaga as letras α; $\rightarrow \mathfrak{B}$.

parâmetros executa a função *copy*, copia o primeiro símbolo marcado com α para o fim da fita, em seguida executa a função *erase*, que encontra o primeiro marcador α e o apaga.
A função de dois parâmetros estende o funcionamento para todos os símbolos da sequência, executando a função de três parâmetros para copiar o símbolo para o fim e remover o primeiro α, após isso, a função de dois parâmetros é chamada, que novamente chama a função de três parâmetros para remover o segundo símbolo e assim sucessivamente.

$\mathfrak{ce}_2(\mathfrak{B},\alpha,\beta)$ $\quad$ $\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B},\beta),\alpha)$
$\mathfrak{ce}_3(\mathfrak{B},\alpha,\beta,\gamma)$ $\quad$ $\mathfrak{ce}(\mathfrak{ce}_2(\mathfrak{B},\beta,\gamma),\alpha)$

$\mathfrak{ce}_3(\mathfrak{B},\alpha,\beta,\gamma)$. The machine copies down at the end first the symbols marked α, then those marked β, and finally those marked γ; it erases the symbols α, β, γ. [6]

As funções $\mathfrak{ce}_2$ e $\mathfrak{ce}_3$ são extensões da função $\mathfrak{ce}(\mathfrak{B},\alpha)$, copiam duas ou três sequências marcadas com símbolos diferentes para o final da sequência, respectivamente. O processo é realizado de forma sequêncial: primeiro os símbolos marcados com α são copiados para o fim (cada símbolo copiado para o final tem o seu respectivo marcador apagado), em seguida, os símbolos marcados com β são copiados para o fim e assim sucessivamente.

A.5 *REPLACE*

$\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\beta)$ $\quad$ $\mathfrak{f}(\mathfrak{re}_1(\mathfrak{C},\mathfrak{B},\alpha,\beta),\mathfrak{B},\alpha)$
$\mathfrak{re}_1(\mathfrak{C},\mathfrak{B},\alpha,\beta)$ $\quad$ E, Pβ $\quad$ $\mathfrak{C}$

$\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\beta)$. The machine replaces the first α by β and $\rightarrow \mathfrak{C}$ $\rightarrow \mathfrak{B}$ if there is no α. [7]

A função *replace* substitui o primeiro α por β. A função *find* é inicialmente usada para achar o primeiro α, em seguida, a função $\mathfrak{re}_1$ substitui o α pelo β. Caso não tenha α a serem substituídos, a máquina vai para o estado $\mathfrak{B}$.

$\mathfrak{re}(\mathfrak{B},\alpha,\beta)$ $\quad$ $\mathfrak{re}(\mathfrak{re}(\mathfrak{B},\alpha,\beta),\mathfrak{B},\alpha,\beta)$

$\mathfrak{re}(\mathfrak{B},\alpha,\beta)$. The machine replaces all letters α, by β; $\rightarrow \mathfrak{B}$. [8]

[6] Tradução do autor: $\mathfrak{ce}_3(\mathfrak{B},\alpha,\beta,\gamma)$. A máquina copia para o fim primeiro os símbolos marcados com α, depois os marcados com β e finalmente os marcados com γ; ela apaga os símbolos α, β, γ.

[7] Tradução do autor: $\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\beta)$. A máquina troca o primeiro α pelo β e $\rightarrow \mathfrak{C} \rightarrow \mathfrak{B}$ se não tem α.

[8] Tradução do autor: $\mathfrak{re}(\mathfrak{B},\alpha,\beta)$. A máquina troca todas as letras α por β; $\rightarrow \mathfrak{B}$.

A versão de três parâmetros da função *replace* substitui todos os α por β. A execução é feita de forma recursiva, sucessivamente chamando a função $\mathfrak{re}$ de quatro parâmetros para substituir cada símbolo, método similar usado nas funções *erase* e *copy e erase* de dois parâmetros.

A.6 *COPY AND REPLACE*

$$\begin{array}{ll} \mathfrak{cr}(\mathfrak{C},\mathfrak{B},\alpha) & \mathfrak{c}(\mathfrak{re}(\mathfrak{C},\mathfrak{B},\alpha,\alpha),\mathfrak{B},\alpha) \\ \mathfrak{cr}(\mathfrak{B},\alpha) & \mathfrak{cr}(\mathfrak{cr}(\mathfrak{B},\alpha),\mathfrak{re}(\mathfrak{B},\alpha,\alpha),\alpha) \end{array}$$

$\mathfrak{cr}(\mathfrak{B},\alpha)$ differs from $\mathfrak{ce}(\mathfrak{B},\alpha)$ only in that the letters α are not erased. The *m*-configuration $\mathfrak{cr}(\mathfrak{B},\alpha)$ is taken up when no letters "α" are on the tape. [9]

A função *copy and replace* de dois parâmetros têm funcionamento similar à função *copy and erase* de dois parâmetros, sendo o único diferencial que em $\mathfrak{cr}$ os marcadores α não são apagados.

A.7 *COMPARE*[10]

$$\begin{array}{lll} \mathfrak{cp}(\mathfrak{C},\mathfrak{A},\mathfrak{E},\alpha,\beta) & & \mathfrak{f}'(\mathfrak{cp}_1(\mathfrak{C}_1\mathfrak{A},\beta),\mathfrak{f}(\mathfrak{A},\mathfrak{E},\beta),\alpha) \\ \mathfrak{cp}_1(\mathfrak{C},\mathfrak{A},\beta) & \gamma & \mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C},\mathfrak{A},\alpha),\mathfrak{A},\beta) \\ \mathfrak{cp}_2(\mathfrak{C},\mathfrak{A},\gamma)\begin{cases} \\ \\ \end{cases} & \gamma & \mathfrak{C} \\ & \text{not } \gamma & \mathfrak{A}. \end{array}$$

The first symbol marked α and the first marked β are compared. If there is neither α nor β, $\rightarrow \mathfrak{E}$. If there are both and the symbols are alike, $\rightarrow \mathfrak{C}$. Otherwise $\rightarrow \mathfrak{A}$. [11]

A função *compare* compara se o primeiro símbolo marcado com α é igual ao primeiro símbolo marcado com β. Primeiramente a função $\mathfrak{f}'$ procura pelo primeiro α, se achar, o cabeçote é movido para a esquerda e passa para a função $\mathfrak{cp}_1$. Seguindo para $\mathfrak{cp}_1$, o símbolo a ser comparado está representado na tabela como um γ, e a máquina passa a procurar pelo primeiro β, ao achar,

[9] Tradução do autor: $\mathfrak{cr}(\mathfrak{B},\alpha)$ difere de $\mathfrak{ce}(\mathfrak{B},\alpha)$ apenas em que no primeiro as letras α não são apagadas. A configuração de máquina $\mathfrak{cr}(\mathfrak{B},\alpha)$ é retomada quando não há letras "α" na fita.

[11] Tradução do autor: O primeiro símbolo marcado com α e o primeiro símbolo marcado com β são comparados. Se não há α nem β, $\rightarrow \mathfrak{E}$. Se tem os dois e os símbolos são iguais, $\rightarrow \mathfrak{C}$. Caso contrário $\rightarrow \mathfrak{A}$.

[11] Como apontado por (PETZOLD, 2008), há dois erros de digitação cometidos por Turing na definição desta função: a primeira é na primeira linha, na terceira coluna, com a ausência da vírgula após o $\mathfrak{C}_1$; a segunda é o uso desnecessário do ponto final na tabela.

o cabeçote é movido para a esquerda e passa para a função $\mathfrak{cp}_2$. Por fim, em $\mathfrak{cp}_2$ os símbolos são comparados, se γ apontado pelo cabeçote é igual ao α, então os dois símbolos são iguais e a configuração final passa a ser $\mathfrak{C}$. Caso não tenha símbolos marcados com α ou β, a configuração final passa a ser $\mathfrak{E}$.

A.8 *COMPARE AND ERASE*[12]

$$\mathfrak{cpe}(\mathfrak{C},\mathfrak{A},\mathfrak{E},\alpha,\beta) \qquad \mathfrak{cp}(\mathfrak{e}(\mathfrak{e}(\mathfrak{C},\mathfrak{C},\beta),\mathfrak{C},\alpha),\mathfrak{A},\mathfrak{E},\alpha,\beta)$$

$\mathfrak{cpe}(\mathfrak{C},\mathfrak{A},\mathfrak{E},\alpha,\beta)$ differs from $\mathfrak{cp}(\mathfrak{C},\mathfrak{A},\mathfrak{E},\alpha,\beta)$ in that in the case when there is similarity the first α and β are erased.

$$\mathfrak{cpe}(\mathfrak{A},\mathfrak{E},\alpha,\beta) \qquad \mathfrak{cpe}(\mathfrak{cpe}(\mathfrak{A},\mathfrak{E},\alpha,\beta),\mathfrak{A},\mathfrak{E},\alpha,\beta).$$

$\mathfrak{cpe}(\mathfrak{A},\mathfrak{E},\alpha,\beta)$. The sequence of symbols marked α is compared with the sequence marked β. $\rightarrow \mathfrak{C}$ if they are similar. Otherwise $\rightarrow \mathfrak{A}$. Some of the symbols α and β are erased. [13]

A função *compare and erase* de cinco parâmetros tem funcionamento similar ao da função *copy* de cinco parâmetros, a diferença está que em que no primeiro, quando os dois primeiros símbolos marcados são iguais, os marcadores α e β são removidos. A função $\mathfrak{cpe}$ de quatro parâmetros compara duas sequências de símbolos marcados com α e β, respectivamente, se ambas as sequências forem iguais, os dois marcadores são removidos. De forma similar às versões recursivas de *copy and replace* e *copy and erase*, a versão de quatro parâmetros de $\mathfrak{cpe}$ usa recursão para percorrer as sequências.

A.9 *FIND*

$$\begin{array}{rlll}
\mathfrak{q}(\mathfrak{C}) & \text{Any} & \text{R} & \mathfrak{q}(\mathfrak{C}) \\
\left\{\right. & \text{None} & \text{R} & \mathfrak{q}_1(\mathfrak{C}) \\
\mathfrak{q}_1(\mathfrak{C}) & \text{Any} & \text{R} & \mathfrak{q}(\mathfrak{C}) \\
\left\{\right. & \text{None} & & \mathfrak{C} \\
\mathfrak{q}(\mathfrak{C},\alpha) & & & \mathfrak{q}(\mathfrak{q}_1(\mathfrak{C},\alpha)) \\
\mathfrak{q}_1(\mathfrak{C},\alpha) & \alpha & & \mathfrak{C} \\
\left\{\right. & \text{not } \alpha & \text{L} & \mathfrak{q}_1(\mathfrak{C},\alpha)
\end{array}$$

[13] Tradução do autor: $\mathfrak{cpe}(\mathfrak{A},\mathfrak{E},\alpha,\beta)$. A sequência de símbolos marcados com α é comparada com a sequência marcada com β. $\rightarrow \mathfrak{C}$ se elas são similares. Caso contrário $\rightarrow \mathfrak{A}$. Alguns dos símbolos α e β são apagados.

[13] Novamente, há um erro de digitação nesta tabela: o uso desnecessário do ponto final na tabela da definição da função $\mathfrak{cpe}$ de quatro parâmetros.

$\mathfrak{q}(\mathfrak{C}, \alpha)$. The machine finds the last symbol of form $\alpha. \rightarrow \mathfrak{C}$. [14]

Enquanto a função $\mathfrak{f}$ acha o símbolo marcado mais à esquerda, $\mathfrak{q}$ acha pelo símbolo mais à direita na sequência, ou seja, o último. A função $\mathfrak{q}$ de um parâmetro termina a computação ao encontrar dois espaços vazios seguidos, caso contrário, a tarefa é repetida até serem encontrados. A função de dois parâmetros chama a função de um parâmetro para chegar ao final da sequência, ao chegar, a função $\mathfrak{q}_1$ é chamada, que vai voltando o cabeçote para a esquerda até encontrar o α.

[14] Tradução do autor: $\mathfrak{q}(\mathfrak{C}, \alpha)$. A máquina acha o último símbolo da forma $\alpha. \rightarrow \mathfrak{C}$.

APÊNDICE B – ARTIGO DO TCC

Apresentação da demonstração da equivalência entre Computabilidade e Lambda-Definibilidade
Um estudo do trabalho de Alan Turing

Caique R. Marques[1], **Jerusa Marchi**[1]

[1] Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)

c.r.marques@grad.ufsc.br, jerusa.marchi@ufsc.br

Resumo. *Em 1928, David Hilbert e Wilhelm Ackermann propuseram um problema para a comunidade matemática da época: sobre a possibilidade de existir algum tipo de procedimento que diga se uma dada fórmula em lógica de primeira ordem é provável ou não. Duas respostas para esse problema vieram, apresentando a negativa, destas duas soluções a primeira foi proposta por Alonzo Church com o uso do Cálculo Lambda [Church 1936], a segunda veio de Alan Turing com o uso de máquinas de computar [Turing 1937b].*
Ambos os trabalhos foram feitos de forma independente, Turing viria a adicionar um apêndice em sua publicação sobre a equivalência dos dois modelos. Pouco depois, ele apresentaria um artigo detalhando mais sobre a prova [Turing 1937a]. Esta última publicação é o objeto de estudo deste trabalho, que tem como objetivo investigar cada ponto relacionado às Máquinas de Turing e Cálculo λ na publicação, como também estudar os conceitos fundamentais de cada modelo.

1. Introdução

Dentre os modelos teóricos existentes na teoria da computação, dois serão o nosso objeto de estudo: as máquinas de Turing e o Cálculo Lambda. Ambos foram desenvolvidos para resolver um problema vigente na época, um problema de decisão. Na tentativa de formalização da matemática, David Hilbert propôs, em 1900, uma lista de vinte e três problemas para debater sobre os problemas vigentes [Hilbert 1902]. Em 1928, Hilbert e Wilhelm Ackermann propuseram um outro problema: eles pediam um algoritmo que fosse capaz de decidir sobre uma sentença válida de lógica de primeira ordem, ou seja, se o algoritmo seria capaz de dizer se a dada sentença lógica é provável ou não [Hilbert and Ackermann 1950]. O problema proposto é chamado de *Entscheidungsproblem* ("problema de decisão" em alemão). Um problema de decisão pode ser definido como uma função cuja saída é uma de duas possibilidades: "sim" ou "não" [Kozen 1997], tendo esta propriedade, o problema é dito *decidível*, caso contrário, é dito *indecidível*.

A primeira solução do problema foi proposta por Alonzo Church, em 1936, apresentando que não existe tal algoritmo [Church 1936]. A proposta de Church foi com o Cálculo Lambda, modelo formal inicialmente proposto para formular uma teoria geral das funções para, assim, fornecer as bases para a matemática e a lógica [Barendregt 1984], no entanto, Church passou a focar mais na parte de formulação da teoria geral das funções o que culminou com a tentativa de formar a definição de efetivamente calculável aplicado a

Cálculo Lambda, chamado de λ-definibilidade. Assim, algo que é calculável por Cálculo Lambda é dito λ-definível.

Pouco depois e independente a Church, Alan Turing propôs uma outra solução ao problema de Hilbert e Ackermann, também apresentando a resposta negativa [Turing 1937b]. Turing propôs um modelo de máquinas de computar, que mais tarde seriam chamadas de Máquinas de Turing, por uma série de estados ou configurações a máquina computaria problemas. A definição de efetivamente calculável em Máquinas de Turing é, portanto, computabilidade. Turing adicionou um apêndice à sua publicação mostrando que tanto as máquinas de computar quanto Cálculo Lambda são equivalentes. Pouco depois da publicação [Turing 1937b], Turing escreveu uma outra detalhando a equivalência entre os dois modelos formais [Turing 1937a].

O objetivo deste trabalho é investigar e apresentar os dois modelos teóricos, computabilidade e λ-definibilidade, e a equivalência de ambos conforme apresentado por Turing na sua publicação de 1937.

2. Cálculo Lambda

2.1. Função matemática

Segundo [Stewart 2008], uma função matemática "é uma regra que associa cada elemento de x de um conjunto D a exatamente um elemento, chamado de $f(x)$, em um conjunto E", portanto, cada elemento x do domínio é associado a um elemento $f(x)$ do contradomínio. Em outras palavras, podemos descrever uma função matemática f como uma "caixa preta" que dada a entrada x, retorna uma saída $f(x)$, conforme ilustrado na figura 1.

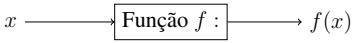


Figura 1. Ilustração de uma função matemática.

2.2. λ-definibilidade

O Cálculo Lambda é um sistema formal criado por Alonzo Church com o propósito de promover uma teoria geral das funções [Church 1936]. Tudo em Cálculo Lambda são funções, assim, o modelo preza pela implementação de regras genéricas que realizam o cálculo através dos parâmetros especificados como entrada, assim, o Cálculo Lambda possui o paradigma funcional. Na computação, a forma de classificar as linguagens formais é chamada de paradigmas da programação, existindo dois tipos: as funcionais e as imperativas [O'Donnel 1977]. Sendo uma linguagem, o Cálculo Lambda possui uma gramática, apresentando um conjunto de produção de regras usando uma sequência de símbolos válidos e são de acordo com a sintaxe da linguagem. A seguir é apresentada a gramática do Cálculo Lambda, adaptada de [Pierce 2002].

$$
\begin{aligned}
&<expressão> ::= \\
&\quad <variável> \\
&\quad \mid \lambda\ <variável>.<expressão> \\
&\quad \mid <expressão>\ <expressão>
\end{aligned}
$$

Uma expressão em Cálculo Lambda pode ser uma de três possibilidades: uma variável, tal como x, y, etc.; uma abstração, que é uma função lambda "regular", composta do símbolo lambda (λ) acompanhado da variável que será atribuída a uma expressão; uma aplicação, que é composta por duas expressões. Assim, definimos problemas que são calculáveis por Cálculo Lambda como problemas λ-definíveis.

2.3. Conversões

Uma expressão de cálculo lambda pode passar por transformações até resultar em sua forma normal, ou seja, até onde não há mais transformações possíveis a serem feitas. Existem três regras de transformações listadas a seguir [Petzold 2008], também ditas como "conversões", onde uma conversão é simbolizada com o uso da expressão $conv$.

1. É possível alterar uma variável ligada se a nova variável não interferir com nenhuma outra coisa na fórmula.
   Exemplo, $\lambda x[x^3 - x + 2](A)\ conv\ \lambda y[y^3 - y + 2](A)$;
2. Na expressão do tipo $\{\lambda x.M\}(N)$, se N não possui nada nomeado como x, é possível substituir N por todas as ocorrências de x em M, ou seja, aplicar uma redução β.
   Exemplo: $\lambda x[x^2 + 2x + 17](A)\ conv\ A^2 + 2A + 17$;
3. O reverso de 2 é permitido.
   Exemplo: $A^2 + 2A + 17\ conv\ \lambda x[x^2 + 2x + 17](A)$.

3. Máquinas de Turing

A Máquina de Turing é um modelo teórico proposto por Alan Turing [Turing 1937b] de uma máquina de computar. A estrutura de uma máquina de Turing pode ser composta dos seguintes componentes: fita, cabeçote e o controle finito. A fita é infinita e composta por campos ou espaços, qualquer coisa que necessitar ser salvo deve ser escrita na fita, portanto, as entradas e as saídas de problemas a serem resolvidos por uma máquina são escritas na fita; o cabeçote pode ser movido para a esquerda e para a direita e aponta para um campo da fita, indicando o símbolo a ser lido ou o espaço a ser escrito; o controle finito corresponde aos estados ou configurações da máquina e qual é o vigente, o processo de computação é definido pela sequência de estados. A figura 2 ilustra uma máquina de Turing.

3.1. Computação de problemas

As máquinas computam problemas, assim, realizam uma série de passos seguindo um modelo bem-definido, como um algoritmo. Os resultados de uma computação depende do tipo do problema especificado, logo, a saída pode ser uma de duas possibilidades: uma resposta do tipo "sim" ou "não", logo, a máquina consegue decidir sobre o problema; um resultado associado à entrada, assim, a máquina é uma função, podendo conter uma série de respostas possíveis de acordo com o que o foi designado de entrada para ela.

Em questão sobre a máquina decisora, havendo uma resposta do tipo "sim" ou "não", então a máquina foi capaz de *decidir* sobre o problema; caso contrário, a máquina não chegue a nenhuma resposta, ela ficará executando o problema eternamente. Logo, os problemas em que resultam numa resposta exata, são ditos problemas *decidíveis*; os problemas em que a máquina é capaz de processar, mas não retornam nenhuma resposta específica, são problemas *reconhecíveis*. Observando que todo problema decidível é reconhecível, mas o contrário não necessariamente é verdadeiro.

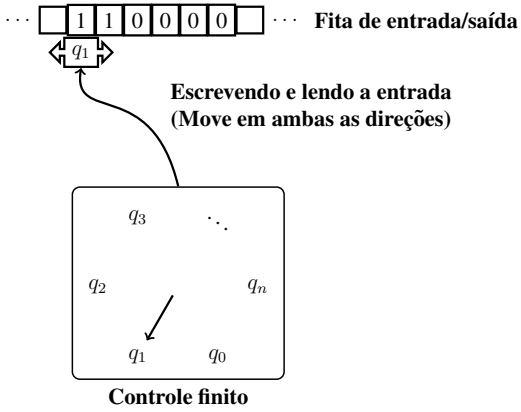


Figura 2. Ilustração de uma máquina de Turing.

4. Demonstração

Nesta seção será apresentada os passos usados por Turing em sua publicação de 1937 [Turing 1937a] para a prova da equivalência entre os modelos Cálculo Lambda e Máquinas de Turing.

4.1. Skeleton tables

Conforme apresentado em [Turing 1937b], a ferramenta usada para programar as máquinas de computar para realizar tarefas é com o uso das *skeleton tables*, que são tabelas que listam o que cada configuração da máquina deve fazer, dependendo de sua entrada. Uma *skeleton table* é composta por quatro colunas: *m-config* ou configuração da máquina, símbolo que será lido pela máquina, operação a se realizar e configuração final de máquina ou próxima configuração. A figura 4 ilustra um exemplo de uma *skeleton table*.

Antes de seguirmos com o estudo do exemplo da *skeleton table*, uma observação é necessária: ao escrever uma sequência de símbolos na fita, Turing separa cada elemento por um espaço em branco, isto é feito para fins de marcação, para que seja mais fácil trabalhar em cima da sequência, logo, se ao percorrer uma sequência chega-se a dois espaços em branco consecutivos (considerando que não há marcações), significa que chegou ao fim da sequência. A figura 3 ilustra um exemplo de uma sequência na fita.

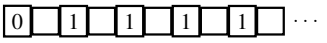


Figura 3. Exemplo da sequência `01111` escrita na fita.

Na tabela ilustrada na figura 4 representa uma função de busca de um elemento x na fita, adaptada da mesma tabela que Turing fez na publicação [Turing 1937b]. Começando na configuração inicial ƒ, é verificado qual o símbolo que o cabeçote está

apontando: se for um xevá (ə), o cabeçote é movido para a esquerda e a próxima configuração será $\mathfrak{f}_1$; caso o cabeçote esteja apontando para outro símbolo qualquer não vazio que não seja um xevá, o cabeçote é movido para a esquerda e continua na mesma configuração $\mathfrak{f}$, portanto, a máquina fica num *loop* até encontrar um xevá. O xevá representa, aqui, o começo da fita.

m-config	*símbolo*	*operações*	*m-config final*
$\mathfrak{f}$	ə	L	$\mathfrak{f}_1$
	não ə	L	$\mathfrak{f}$
$\mathfrak{f}_1$	x		$\mathfrak{q}$
	não x	R	$\mathfrak{f}_1$
	Nada	R	$\mathfrak{f}_2$
$\mathfrak{f}_2$	x		$\mathfrak{q}$
	não x	R	$\mathfrak{f}_1$
	Nada	R	$\mathfrak{r}$

Figura 4. *Skeleton table* referente à função de busca $\mathfrak{f}$.

Na configuração $\mathfrak{f}_1$ há três possibilidades: se o símbolo apontado pelo cabeçote é o x, então o símbolo desejado foi encontrado e a computação é terminada, com a configuração final da máquina sendo $\mathfrak{q}$, indicando tal feito; se o símbolo apontado pelo cabeçote for qualquer outro símbolo não vazio que não é x, o cabeçote move para a direita e a máquina continua em $\mathfrak{f}_1$; se o cabeçote estiver apontando para um símbolo vazio, então o cabeçote é movido para a direita e a configuração da máquina passa a ser $\mathfrak{f}_2$.

Na configuração $\mathfrak{f}_2$ novamente temos três possibilidades: caso o símbolo x tenha sido encontrado, a computação termina com a máquina na configuração final $\mathfrak{q}$; se o cabeçote está apontando para um símbolo qualquer não vazio que não é x, então o cabeçote é movido para a direita e a máquina volta para a configuração $\mathfrak{f}_1$; por fim, caso o cabeçote esteja apontando para um espaço vazio, segundo consecutivo apontado desta vez, então significa que a máquina chegou ao fim da sequência e, portanto, não achou o símbolo x, o cabeçote é movido para a direita e a configuração final da máquina passa a ser $\mathfrak{r}$.

4.2. Cálculo $\lambda - K$

Na subseção anterior foi apresentado o método que Turing usou para programar a máquina, com o uso de *skeleton tables*. Nesta seção será expressada sobre uma modificação do Cálculo λ, para ser mais fácil de trabalhar em máquinas, o Cálculo $\lambda - K$.

No geral, o Cálculo $\lambda - K$ é similar ao Cálculo λ, porém, com algumas restrições são impostas para o primeiro:

1. Uso exclusivo de colchetes, sem usos de parênteses ou chaves;
2. As variáveis serão definidas com o símbolo x e, se necessitar, do símbolo $'$, por exemplo x, x', x'', etc.. O símbolo $'$ pode ser repetido quantas vezes for necessário. Sem qualquer uso de outra letra latina, tais como a, b, y, etc.;
3. Mudança na forma da transformabilidade, no caso, as regras de conversão não são alteradas em si, apenas no sentido lexical, de acordo com os dois itens especificados anteriormente.

Dadas as restrições, o item 3 mostra que uma modificação nas regras de conversão é necessária, portanto, em seguida Turing define as três regras de conversão para Cálculo $\lambda - K$, que são apresentadas a seguir:

1. M é da forma $\lambda V[X]$ e N é da forma $\lambda U[Y]$ onde Y é obtido de X trocando a variável V pela variável U, sendo que não haja ocorrências de U em X;
2. M é da forma $[\lambda V[X]][Y]$ onde V é uma variável e N é obtido substituindo V por Y em X. Existem restrições quanto à este item, porque se W for a variável V ou uma variável em Y, então não pode haver λW em X;
3. N é imediatamente transformável em M como citado no item 2.

Notadamente, as regras de conversão do Cálculo $\lambda - K$ são similares às regras de conversão do Cálculo λ, como apresentado na subseção 2.3 - o diferencial está no uso de símbolos permitidos, onde o Cálculo $\lambda - K$ é mais restrito neste aspecto, o que facilita para a computação na máquina, no entanto, as expressões em Cálculo $\lambda - K$ serão mais verbosas do que as do Cálculo λ. Para ilustrar um exemplo, Turing define um número zero e uma função sucessora S. O número zero é expandido como:

$$\lambda x[\lambda x^I[x^I]],$$

e a função sucessora S é definida da seguinte forma:

$$\lambda x^{II}[\lambda x[\lambda x^I[[x][[[x^{II}][x]][x^I]]]]].$$

O resultado da execução da função $[S][0]$ deve resultar, portanto, numa função $\lambda - K$ cuja a redução é o número 1. Assim, substituindo as respectivas fórmulas, temos:

$$\begin{aligned}
&[S][0] \\
&= \mathbf{[}\lambda x^{II}[\lambda x[\lambda x^I[[x][[[x^{II}][x]][x^I]]]]]\mathbf{]}\mathbf{[}\lambda x[\lambda x^I[x^I]]\mathbf{]} \\
&= \lambda x[\lambda x^I[[x][[\mathbf{[}\boldsymbol{\lambda x[\lambda x^I[x^I]]][x]}\mathbf{]}[x^I]]]] \\
&= \lambda x[\lambda x^I[[x][[\mathbf{[}\boldsymbol{\lambda x^I[x^I]]][x^I]}\mathbf{]}]]] \\
&= \lambda x[\lambda x^I[[x][[[x^I]]]]] \\
&= 1.
\end{aligned}$$

A função $[S][1]$ irá resultar na função $\lambda - K$ "$\lambda x[\lambda x^I[[x][[x][x^I]]]]$" que é a expansão do número 2 e assim sucessivamente.

4.3. Abreviações

Nas subseções anteriores foram apresentadas como é o método de computação da máquina e uma versão mais restrita, lexicalmente falando, do Cálculo λ, chamada de Cálculo $\lambda - K$. Nesta subseção será apresentado o funcionamento de algumas tabelas complementares.

Em [Turing 1937b], Turing criou diversas *skeleton tables* para lidar com a computação do problema de decisão, em [Turing 1937a], na seção de abreviações, o autor adicionou tabelas complementares às que ele havia criado antes, as novas tabelas lidam com sequências ao invés de um símbolo só. Turing também chama as *skeleton tables*

como abreviações, porque segundo ele as tabelas não são essenciais, no entanto, servem para facilitar o entendimento do funcionamento da máquina, além da facilidade de construção [Turing 1937b].

São dez novas tabelas especificadas nesta seção, sendo quatro complementares e todas elas usam tabelas já criadas em [Turing 1937b] em suas implementações. As tabelas e os respectivas descrições serão listadas a seguir.

- *Print at the end and mark* ($\mathfrak{pem}(\mathfrak{A}, \alpha, \beta)$): Em uma sequência de símbolos, a máquina imprime α no fim da sequência e marca com β. Esta função complementa $\mathfrak{pe}(\mathfrak{C}, \beta)$, que imprime β no fim de uma sequência de símbolos;
- *Copy and erase and mark* ($\mathfrak{cem}(\mathfrak{A}, \gamma, \beta)$): Copia uma sequência de símbolos marcados com γ para o fim da fita, cada símbolo desta nova sequência é marcado com β e os marcadores γ são apagados. Esta função é complemento de $\mathfrak{ce}(\mathfrak{A}, \gamma)$, que realiza o mesmo procedimento, copia a sequência para o fim e apaga os marcadores γ, porém, a sequência resultante não é marcada com símbolo algum;
- *Copy and replace and mark* ($\mathfrak{crm}(\mathfrak{A}, \gamma, \mathfrak{B})$): Similar à função $\mathfrak{cem}(\mathfrak{A}, \gamma, \beta)$ do item anterior, o diferencial é que os marcadores γ não são apagados. Complemento da função $\mathfrak{cr}(\mathfrak{A}, \gamma)$ e é similar, copia a sequência marcada com γ para o fim da fita, porém, a sequência resultante não é marcada com símbolo algum;
- *Compare* ($\mathfrak{cpr}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta)$): A sequência de símbolos marcada com α é comparada com a sequência marcada com β. Se ambas forem iguais, então a configuração final da máquina é $\mathfrak{A}$, caso contrário, é $\mathfrak{C}$. Esta função é complemento de $\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$, que compara o primeiro símbolo marcado com α com o primeiro símbolo marcado com β: se não tiver símbolos marcados com α nem β, então a configuração final da máquina é $\mathfrak{C}$; se tiverem tais símbolos e se eles forem iguais, então a configuração final da máquina é $\mathfrak{A}$; caso os tais símbolos não sejam iguais, então a configuração final é $\mathfrak{E}$;
- *Brackets* ($\mathfrak{bt}(\mathfrak{A}, \alpha, \beta, \gamma, \delta)$): Descreve a forma de encontrar pares de colchetes. Observando os parâmetros da função, $\mathfrak{A}$ é a configuração final da máquina após a computação da função, α representa o símbolo de abre colchetes ([), β representa o símbolo de fecha colchetes (]), a análise de encontrar pares de colchetes é realizada sobre a sequência de símbolos marcada com γ e δ é a marcação usada para símbolos dentro de um par de colchetes;
- *Substitution* ($\mathfrak{sub}(\mathfrak{A}, \alpha, \beta, \gamma, \delta)$): Esta função resulta na substituição da sequência marcada com γ pela sequência marcada com β na sequência marcada com α. Considerando que a sequência `100` esteja marcada com γ, `011` com α e `1` com β, então o resultado da função será `0100100`, ou seja, os `1` da sequência α foram substituídos por `100`;
- $\mathfrak{dt}(\mathfrak{A}, \mathfrak{B}, \alpha, \beta)$: Verifica se a sequência de símbolos marcados com β é subsequência da sequência de símbolos marcados com α. A configuração da máquina será $\mathfrak{A}$ caso afirmativo, $\mathfrak{B}$ caso contrário.

As três tabelas finais definidas nesta seção são referentes a enumeração, assim, relacionadas a tomadas de decisão. Em tabelas posteriores haverão casos que uma decisão deverá ser tomada e, dependendo da decisão, o resultado difere, por isso a necessidade de enumerar os possíveis caminhos que uma dada computação pode chegar - a forma de enumeração será definida usando dois símbolos, sim ou não, 0 ou 1, verdadeiro ou falso.

Turing usa as letras L e M para enumerar, nas funções, L ou 0 é representado por ζ e M ou 1 é representado por η.

- *Addition* ($\mathfrak{add}(\mathfrak{A}, \alpha, \zeta, \eta)$): Acrescenta mais na sequência de símbolos marcados com α, que é composto por ζ e η. Por exemplo, se a sequência for `00`, então o resultado de $\mathfrak{add}$ será `10`; se a sequência for `110`, então o resultado de $\mathfrak{add}$ será `001`;
- *Choice* ($\mathfrak{ch}(\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \alpha, \zeta, \eta)$): Uma escolha é realizada, onde a sequência de símbolos marcados com α é composta por ζ e η. Se o primeiro símbolo não usado da sequência for ζ, então a configuração final da máquina será $\mathfrak{A}$; se o primeiro símbolo não usado for η, então a configuração final será $\mathfrak{B}$; caso todos os símbolos da sequência α já foram usados, então todos os símbolos usados são marcados novamente com α para serem reusados e a configuração final da máquina será $\mathfrak{C}$;
- $\mathfrak{cch}(\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \alpha, \zeta, \eta)$: Similar à função do item anterior, o diferencial é que ao terminar a computação desta função, o cabeçote volta para a posição onde estava antes de chamar $\mathfrak{cch}$.

4.4. Conversão mecânica

Até então foram apresentados a forma de escrever tabelas para a computação da máquina, a definição de uma variante do Cálculo λ para trabalhar com máquinas, o Cálculo $\lambda - K$, a complementação de tabelas existentes e a criação de novas para ajudar a lidar com as operações. Nesta subseção será mostrada a descrição das tabelas para as regras de conversão do Cálculo $\lambda - K$.

São cinco tabelas referentes às regras de conversão do cálculo $\lambda - K$, sendo três são implementações de cada uma das três regras de conversão, apresentadas em 4.2.

- $\mathfrak{pff}(\mathfrak{A}, \mathfrak{C}, \alpha, \theta)$: Referente à terceira regra da conversão. A função faz uso da sequência de enumeração, com os símbolos ζ e η e também é considerado que as expressões $\lambda - K$ a serem trabalhadas estão escritas na fita e estão separadas por ponto e vírgula;
- $\mathfrak{va}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta, \theta)$: Referente à primeira regra da conversão. A expressão $\lambda - K$ a ser trabalhada corresponde à sequencia de símbolos marcados com α e o resultado é marcado com β;
- $\mathfrak{red}(\mathfrak{A}, \alpha, \beta)$: Referente à segunda regra da conversão. A expressão $\lambda - K$ a ser trabalhada corresponde à sequência de símbolos marcados com α e o resultado é marcado com β;
- $\mathfrak{imc}(\mathfrak{A}, \mathfrak{C}, \alpha, \beta, \theta)$: Realiza uma conversão imediata, ou seja, realiza uma conversão na expressão $\lambda - K$, que corresponde à sequência de símbolos marcados com α. O resultado é marcado com β;
- $\mathfrak{conv}(\mathfrak{A}, \alpha, \beta, \theta)$: Realiza conversões imediatas de uma dada expressão $\lambda - K$, que é uma sequência de símbolos marcados com α, até resultar na forma normal. O resultado é marcado com β.

4.5. Computabilidade das funções λ-definíveis

Nas subseções anteriores foi apresentado como é definido as tabelas para a computação da máquina, a definição de um modelo mais restrito lexicalmente do Cálculo λ, o Cálculo $\lambda-$

K, o complemento de tabelas antes definidas e a criação de novas e, até então, definição de tabelas para as operações de conversão do Cálculo $\lambda - K$. Agora que todos os recursos da máquina foram especificados, Turing apresenta a tabela final que mostra que uma máquina de Turing pode computar Cálculo $\lambda - K$. Primeiramente, Turing relembra o conceito de computabilidade, que é o mesmo que ele usou na publicação [Turing 1937b] em relação ao problema de decisão:

> [...] if $f(n)$ is $\lambda - K$-definable then the sequence γ_f in which there are $f(n)$ figures 1 between the nth and the $(n+1)$th 0, and $f(0)$ figures before the first 0, is computable [Turing 1937a, p. 160].[1]

A máquina teria que ser capaz, portanto, de executar cada possibilidade de uma dada função $\lambda - K$, escrevendo todos os resultados que seriam compostos na sequência γ_f, cada resultado seria separado por 1. Um exemplo para ilustrar a definição, dada a seguinte função:

$$f(x) = 2x + 1.$$

Como já dito em 2.1, uma função matemática relaciona um elemento do domínio com o contradomínio. Assim se aplicarmos alguns exemplos à função $f(x)$ teremos:

$$\begin{aligned} f(0) &= 2.0 + 1 = 1 \\ f(1) &= 2.1 + 1 = 3 \\ f(2) &= 2.2 + 1 = 5 \\ f(3) &= 2.3 + 1 = 7. \end{aligned}$$

A máquina que seria capaz de computar a função $f(x)$ imprimiria o número de zeros correspondente aos resultados acima, cada resultado separado por 1. Assim, a máquina deve escrever:

$$\texttt{0100010000010000000},$$

que seria a sequência γ_f.

A seguir é apresentado, em forma de pseudo-código, a *skeleton table* referente à computação de funções $\lambda - K$. A função F corresponde à função $\lambda - K$ que está sendo trabalhada e S representa a função sucessora, como apresentada na subseção 4.2.

[1] Tradução do autor: se $f(n)$ é $\lambda - K$-definível então a sequência γ_f em que $f(n)$ figura 1 entre o n-ésimo e o $(n+1)$-ésimo 0 e $f(0)$ figura antes do primeiro 0, é computável.

```
1: h ← F
2: i ← 0
3: k ← i
4: u ← S
5: d ← i
6: v ← [h][k]
7: if d ≠ i then
8:     d ← [u][d]
9: m ← d
10: w ← conv(v)
11: if w ≠ m then
12:     goto 6
13: while m ≠ 0 do
14:     a.insert(1)
15:     m ← m − 1
16: a.insert(0)
17: m ← ∅, w ← ∅, v ← ∅, s ← ∅
18: k ← [u][k]
19: goto 6
```

As linhas 1 a 5 são a atribuição de variáveis básicas que serão usadas por todo o processo, no caso das tabelas, essa atribuição é similar à marcação dos símbolos de uma sequência, por exemplo, a linha 1 significa que a sequência que corresponde à função F cada símbolo é marcado com h. Na linha 6, a função $[h][k]$, que é $[F][0]$, é marcada com v. Nas linhas 7 e 8 há o condicional para fins de validação, assim, na segunda iteração deste pseudo-código, o valor de d é atualizado. Na linha 10 é realizada a conversão de v, ou seja o resultado de $[F][0]$, que é marcada com w. As duas linhas seguintes novamente são usadas para validação. No *loop* que começa na linha 13 é onde o resultado é escrito e marcado com a, assim, escreverá o número de uns correspondentes ao resultado em m, no fim, um zero é escrito ao fim da sequência para separação do resultado seguinte. Por fim, as variáveis usadas são zeradas, ou seja, os marcadores em questão são apagados e a função $[u][k]$, $[S][0]$ neste caso, é marcado com k. A computação volta para a linha 6 que continua com a função seguinte em k e assim sucessivamente.

Deste modo, Turing construiu uma máquina capaz de computar Cálculo $\lambda - K$ que é equivalente ao Cálculo λ. Como as definições de Cálculo $\lambda - K$ já foram definidas, no caso, conceitos de fórmulas e as regras de convertibilidade, isso reforça a prova da computabilidade do modelo formal pelas máquinas de Turing.

5. Trabalhos futuros

A publicação [Turing 1937a] contém uma última seção apresentando a recursividade das funções computáveis que o autor descreveu, no caso, das tabelas. Esta seção foi omitida neste trabalho por questões de escopo, pois os objetos de estudo deste trabalho foram o Cálculo Lambda e as Máquinas de Turing, no entanto, está aberta a possibilidade de trabalho futuro o estudo das funções recursivas de Gödel-Herbrand, além de estudar a publicação de Kleene [Kleene 1936] sobre a equivalência das funções λ-definíveis e recursivas. Outro tópico de estudo são os problemas de Hilbert propostos em 1900, que foram a motivação da criação dos modelos formais estudados neste trabalho, em [Yandell 2002, p. 385] há uma discussão sobre os vinte e três problemas, mas sem menção ao cancelado vigésimo quarto problema.

6. Conclusão

Neste trabalho investigamos, estudamos e apresentamos cada ponto da publicação [Turing 1937a] em relação às Máquinas de Turing e Cálculo λ; além de estudar e mostrar alguns conceitos fundamentais de cada modelo. Turing desenvolveu o modelo teórico de uma máquina de computar em 1937 a fim de resolver um problema vigente, assim, em 1952, Kleene reformula o modelo de máquinas de Turing, apresentando-os no livro *Introduction to Metamathematics*, onde dentre as modificações, a máquina não contém uma configuração específica para parar a máquina - indo para uma configuração diferente, a máquina para ou retorna o resultado - assim, as máquinas do modelo de Kleene tem o comportamento mais fiel aos computadores de hoje.

O cálculo λ foi proposto para se ter uma teoria geral das funções, assim, dentre os legados está na direta influência nas linguagens de programação. Além da influência direta e notável em linguagens de paradigma funcional, tais como Haskell e Lisp, o cálculo λ também teve influência indireta em linguagens imperativas, como apontado por Peter Landin em [Landin 1965a, Landin 1965b], que apresentou uma correspondência entre expressões em ALGOL 60 e cálculo λ. O ALGOL 60 foi influência de muitas linguagens imperativas subsequentes, como Pascal e C.

Referências

Barendregt, H. P. (1984). *The Lambda Calculus*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. Elsevier.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363.

Hilbert, D. (1902). Lecture delivered before the international congress of mathematicians at paris in 1900. *Bulletin of the American Mathematical Society*, 8(10).

Hilbert, D. and Ackermann, W. (1950). *The Principles of Mathematical Logic*. Chelsea Publishing Company, Nova Iorque, NY, Estados Unidos.

Kleene, S. C. (1936). λ -definability and recursiveness. *Duke Mathematical Journal*, 2(2):340–353.

Kozen, D. C. (1997). *Automata and Computability*. Springer.

Landin, P. J. (1965a). Correspondence between algol 60 and church's lambda-notation: Part i. *Communications of the ACM*, 8(2):89–101.

Landin, P. J. (1965b). A correspondence between algol 60 and church's lambda-notations: Part ii. *Communications of the ACM*, 8(3):158–167.

O'Donnel, M. J. (1977). *Computing in systems described by equations*. Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, Baden-Württemberg, Alemanha.

Petzold, C. (2008). *The Annotated Turing*. Wiley Publishing, Inc, Indianópolis, IN, Estados Unidos.

Pierce, B. C. (2002). *Types and Programming Languages*. The MIT Press, Cambridge, MA, Estados Unidos.

Stewart, J. (2008). *Calculus*. Thomson, Belmont, CA, Estados Unidos, 6ªedition.

Turing, A. M. (1937a). Computability and λ-definability. *The Journal of Symbolic Logic*, 2:153–163.

Turing, A. M. (1937b). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265.

Yandell, B. H. (2002). *The Honor Class*. A K Peters, Ltd., Natick, MA, USA.