VI ERIMG - DCC - UFLA - 29 a 31 de Agosto de 2007

Computabilidade: Um pouco de História ...

... um pouco de Matemática

Isabel Cafezeiro e Edward Hermann Heausler

Resumo

Neste artigo apresentmos o contexto histórico em que se desenvolveram importantes resultados da Teoria da Computabilidade. Apresentamos os Teoremas de Gödel e discutimos suas implicações computacionais.

1. O que é Computabilidade?

> *The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.* Turing, A. (1936)

O termo "computável" foi proposto por Alan Turing para designar a totalidade de números reais cuja expansão decimal pode ser calculada em tempo finito, através de recursos finitos. No mesmo artigo, Turing argumenta que este conjunto corresponde ao conjunto de funções ou predicados computáveis. A proposta de Turing consistia em identificar os possíveis processos envolvidos no ato de "computar um número". Para tanto, Turing definiu um artefato teórico, que chamou de "máquina de computar", de maneira que, todo número cuja expansão decimal pudesse ser obtida a partir de operações da máquina seria chamado de "número computado pela máquina".

> *"What are the possible processes which can be carried out in computing a number?"* Turing, A. (1936)

Ao identificar os processos necessários para computar um número, Turing estaria indiretamente, identificando o conjunto de números (funções, predicados, etc) computáveis, e, conseqüentemente, respondendo a seguinte pergunta: O que pode ser efetuado pela máquina de computar?

> *No attempt has yet been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The arguments which I shall use are of three kinds. a. A direct appeal to intuition. b. A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal). c.Giving examples of large classes of numbers which are computable.* Turing, A. (1936)

Na tentativa de definir o "computável" Turing se deparou com o seguinte problema: Os números computados pela máquina realmente formam aquele conjunto que, intuitivamente, consideramos "computável"? Para responder a tal questão com precisão, Turing precisaria provar matematicamente que ($\Rightarrow$) todos os números que consideramos "computáveis" podem ser obtidos pela máquina. Desta maneira, a máquina computaria necessariamente todos os "números computáveis". Por outro lado seria também necessária a prova matemática de que ($\Leftarrow$) todos os números que a máquina computa são considerados por nós "números computáveis". Estaria então provado que a máquina não computaria nada além do que "números computáveis".

Tomemos como base uma apresentação da Máquina de Turing formulada por Hopcroft and Ullman (1979):

> *Turing machine is a 7-tuple* $< Q , \Gamma , b , \Sigma, \delta, Q_0, F >$ *where Q is a finite set of states;* Γ *is a finite set of the tape alphabet/symbols;* $b \in \Gamma$ *is the blank symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation);* Σ*, a subset of* Γ *not including b is the set of input symbols;* δ*:* $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ *is a partial function called the transition function, where L is left shift, R is right shift;* Q_0 *is the initial state; F is the set of final or accepting states.*

Observe que, a máquina de computar de Turing é uma definição formal: consiste de definições de símbolos ($Q , \Gamma , b , \Sigma, \delta, Q_0, F$, L e R) e regras precisas de manipulação destes símbolos (dada pela definição da função δ*:* $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$). A este aparato formal (a Máquina de Turing) queremos fazer corresponder uma realidade: números, e o processo real de computá-los em nossas mentes, ou nas máquinas que existem fisicamente no mundo real.

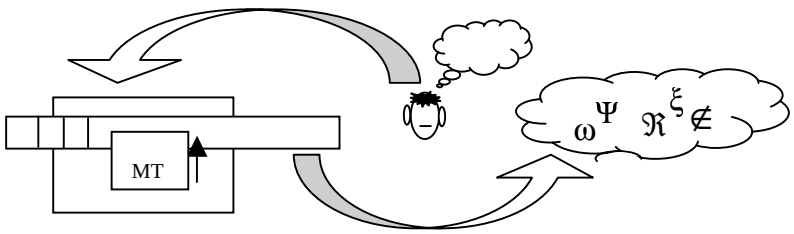


Figura 1: Objeto representado e representação

Cabe questionar:

Como estabelecer a correspondência exata entre a definição formal apresentada e o objeto do mundo real ao qual queremos que esta definição corresponda? A definição apresentada reflete a realidade? A definição apresentada impõe uma nova a realidade?

Em face às dificuldades em estabelecer uma correspondência exata entre o que se quer definir e a definição, Turing assume a tripla estratégia de: apelar à intuição; estabelecer correspondências com outras definições que se proponham a definir o mesmo objeto; e usar exemplos convincentes.

De 1936, quando Turing inicialmente apresentou o conceito "computável", até 2007, podemos observar um enorme avanço tecnológico e científico. Neste novo contexto, podemos agora definir "computável" de maneira comprovadamente fiel à realidade? Persiste a necessidade de apelar à intuição, a exemplos, e outras estratégias? Os problemas considerados não computáveis em 1936 são os mesmos problemas não computáveis dos dias de hoje?

Entendendo "computabilidade" como conceituação de tudo que pode ser realizado por computador a discussão a respeito deste tema é fundamental aos estudantes de Ciência da Computação. A *Computabilidade* vem mostrar ao profissional em formação dois importantes resultados (negativos) com relação ao computador. O primeiro resultado é conseqüência do trabalho de Gödel, e consiste na constatação de que nem tudo se pode resolver através do computador. O segundo resultado negativo, abordado por Turing, aponta a impossibilidade de caracterizar a classe de problemas computáveis. Informalmente, poderíamos dizer que *sabemos que existem problemas não resolvíveis através de computadores, mas não sabemos exatamente quais são.* É através do estudo da *Computabilidade* que o profissional da área de informática adquire pleno conhecimento da real capacidade dos computadores, e passa a lidar com seu instrumento de trabalho de maneira realística, sem mitos.

O Problema da Parada: um exemplo clássico e atual de problema não computável

Escreva um programa que, ao receber um outro programa P juntamente com sua entrada E, pára e imprime "sim" se P, ao receber E, pára, e, pára e imprime "não" se P, ao receber E, entra em *loop*.

Qualquer tentativa para escrever tal programa, excetuando-se os casos triviais, necessariamente implica em alguma estratégia para acompanhar o fluxo de execução de P. Para que a resposta "sim" ou "não" seja emitida, é necessário que esta estratégia forneça uma conclusão em tempo finito. Porém, se o programa entrar em *loop*, a estratégia reproduzirá o *loop*, e a resposta "não" nunca será emitida.

2. Antecedentes Históricos

2.1 A Fundamentação da Matemática

Ao fim do século XIX os matemáticos se depararam com situações que apontavam a necessidade de uma fundamentação mais cuidadosa da Matemática. Uma destas questões diz respeito ao conceito de séries, e pode ser ilustrada pelo paradoxo de Zenon:

> *Aquiles e a Tartaruga decidem apostar uma corrida de 100 metros. Aquiles corre 10 vezes mais rápido do que a tartaruga, e por isto, a tartaruga inicia com 80 metros de vantagem. Aquiles percorre rapidamente a distância inicial que o separa da tartaruga, mas ao alcançar os 80 metros iniciais, a tartaruga já se encontrará 8 metros à frente. Ao alcançar mais 8 metros à frente, a tartaruga já terá avançado mais 0,8 metros, e assim, Aquiles nunca alcançará a tartaruga.*

Este paradoxo leva a crer que a noção de movimento é impossível quando assumimos que tempo e espaço são infinitamente subdivisíveis (Carnielli, W. and Epstein, R. (2006)). Raciocinando desta forma, os matemáticos do século XIX supunham que a soma de infinitos intervalos é infinita. Mas a experimentação mostrava que Aquiles alcançaria a tartaruga em pouco tempo. O confronto da observação dos fatos com a explicação matemática demonstrava que o aparato matemático da época era insuficiente para formalizar alguns fenômenos. A compreensão do paradoxo viria através do conceito de series: os intervalos formam uma progressão geométrica e sua soma converge para um valor finito.

As inquietações dos matemáticos com relação a conceitos não completamente compreendidos e formalizados levaram a grandes avanços neste período. Podemos citar, por exemplo, os seguintes marcos da matemática desta época:

Dedekind (1831-1916) Estabelece o Princípio da Indução e o conceito de número real.

Cauchy (1789-1857) Formula a aritmetização da análise, define limite, funções, funções contínuas e convergência de séries e seqüências infinitas.

Peano (1889) Axiomatiza a Aritmética.

Hilbert (1898-1899) Apresenta a fundamentação da Geometria.

Bolzano concebe a noção abstrata de conjunto finito e infinito.

Cantor (1867 – 1871) Define a Teoria dos Conjuntos e prova a existência de conjuntos infinitos com diferentes cardinalidades.

> *Em 1872, o matemático alemão Dedekind mostra como é possível definir precisamente os números reais a partir dos números inteiros e, em 1888, ele consegue reduzir a aritmética a apenas três conceitos básicos: a existência do número 1, o fato de todo número ter um sucessor e o princípio da indução matemática. Finalmente, em 1889, o matemático italiano Giuseppe Peano estabelece os axiomas da aritmética hoje aceitos, reduzindo o problema da consistência da matemática ao problema da consistência da aritmética, isto é, de seus axiomas.* Bittencourt (2006)

A empolgação com os avanços matemáticos, e particularmente experiências bem sucedidas de explicar conceitos matemáticos através de outros conceitos mais simples conduziram alguns matemáticos a uma abordagem, que, mais tarde ficou conhecida como Abordagem Formalista. Em linhas gerais, os formalistas concebiam a matemática como sistema puramente formal, consistindo na manipulação de símbolos desprovidos de significado ou interpretação.

Empenhados em explicar a matemática através de símbolos, regras precisas, e mecanismos finitários, os formalistas prosseguiram em sua utopia. Neste processo, foram fortemente motivados pela descoberta de paradoxos que demonstravam o perigo da matemática "ingênua", baseada na intuição. Um destes paradoxos é o paradoxo de Russel, que demonstrou a existência de contradição no interior da teoria (ingênua) dos conjuntos. Apresentamos, a seguir, em sua versão informal:

> *O barbeiro de Sevilha faz a barba de todas as (e somente das) pessoas em Sevilha que não fazem a barba a si próprias. Pergunta-se: o barbeiro de Sevilha barbeia-se a si próprio?*

A quem responder afirmativamente a questão acima, argumenta-se: "não é possível, pois o barbeiro de Sevilha não barbeia a quem barbeia a si próprio." A quem responder negativamente a questão, argumenta-se: "não é possível, pois o barbeiro de Sevilha barbeia todos que não barbeiam a si próprio."

A versão matemática do paradoxo consiste em considerar o conjunto $Z = \{X \text{ tal que } X \notin X\}$. Questiona-se: $Z \notin Z$?

Assumindo que $Z \in Z$, temos, pela definição do conjunto Z, que $Z \notin Z$. Por outro lado, assumindo que $Z \notin Z$, visto que todos os conjuntos que não pertencem a si próprio pertencem a Z, temos que $Z \in Z$.

2.2 Interpretação computacional da abordagem formalista

A idéia de considerar a matemática como um sistema formal empolgava os matemáticos do século XIX. Os resultados obtidos naquela época, precedentes à invenção do computador, se aplicam hoje em dia, já que o computador é um sistema formal. É capaz apenas de distinguir a diferença entre dois níveis de tensão, que representamos pelos símbolos 0 e 1, e de efetuar tarefas muito simples de reescrita destes símbolos, como resposta a comandos específicos que lhe fornecemos. Sendo assim, tanto a máquina de computar de Turing (1936), quanto a iniciativa formalista de Hilbert (1900 a 1936), apesar de antecederem cronologicamente a invenção do primeiro computador, dizem respeito aos computadores de hoje em dia.

2.3 Paris, 1900

> *(...) Essa convicção da solubilidade de qualquer problema matemático representa para nós um poderoso estímulo durante o trabalho; escutamos em nós a voz constante: "Esse é o problema, procura a solução. Podes encontrá-la pelo pensamento puro, pois em matemática não existe nenhum ignorabimus!"*. Hilbert, D. (1900)

Em 1900, o matemático alemão David Hilbert lançou, no Segundo Congresso Internacional de Matemática, em Paris, um desafio aos matemáticos da época. Ele reuniu uma lista de 23 problemas em aberto, e convocou uma união de esforços para que se buscasse a solução daqueles problemas. Este episódio é peça relevante na busca pela fundamentação da matemática.

Naquela época, os matemáticos e filósofos se sentiam incomodados com a existência de problemas cuja falsidade ou veracidade, até então, não haviam sido provadas. A presença de problemas supostamente verdadeiros ou supostamente falsos permeando todo aparato matemático representava uma ameaça ao rigor matemático que se buscava. Além disso, não havia consenso com relação a uma parte da matemática identificada por Hilbert como *ideal* ou *abstrata*. Hilbert contrapunha à matemática ideal o conjunto de números naturais e suas manipulações finitárias, ao qual denominava matemática real, e acreditava na matemática como um sistema formal, sustentado por uma pequena quantidade de axiomas, e completo: qualquer proposição expressa naquele sistema poderia ser provada no próprio sistema. Segundo Hilbert, a matemática infinitária (ideal) vinha a ser uma extensão conservativa da matemática finitista, o que significa dizer que a qualquer enunciado demonstrado na matemática ideal tem-se uma prova na matemática real.

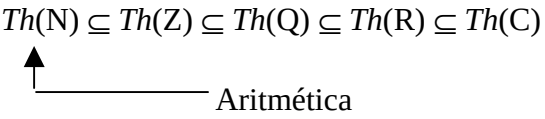


Figura 2: A visão de Hilbert: A aritmética, *Th*(N), é a teoria mais simples. Todas as outras são extensões dela.

2.4 Bolonha, 1928

O *Programa de Hilbert*, lançado em 1928, propunha a formalização da matemática visando garantir rigidez e solidez a toda construção matemática.

Em termos gerais, a proposta era reduzir toda matemática a manipulações reais e responder afirmativamente às seguintes questões. *A matemática é completa? É consistente? É decidível?*

A terceira das questões passou a ser conhecida como "O Problema de Decisão de Hilbert" ("Hilbert's Entscheidungsproblem"): encontrar um mecanismo genérico (e finitário!) que, ao considerar um enunciado qualquer formulado em Lógica de Primeira Ordem, fosse capaz de verificar sua validade ou não.

De 1900 a 1930, grande parte da comunidade matemática mundial acreditou na existência de uma matemática segura, finitária, provadamente correta e livre de imprecisões.

2.5 Viena, 1930/31

> *(...) Em seu diário, Petros assinala a data exata com um comentário lacônico, a primeira e última referência cristã que encontrei em suas anotações: "17 de março de 1933. Teorema de Kurt Gödel. Que Maria, Mãe de Deus, tenha piedade de mim!"*. Doxiadis, A. (2001)

O sonho da matemática sólida e segura teve curta duração. Hilbert, bem como toda comunidade matemática, foi surpreendido em 1930/31 pela publicação dos Teoremas da Incompletude de Gödel. Estes jogaram por terra as duas primeiras questões do programa de Hilbert, e o fez retornar de sua aposentadoria para repensar as bases matemáticas. Os resultados de Gödel publicados em *"Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme"* (Sobre proposições formalmente indecidíveis dos Principia Mathematica e outros sistemas semelhantes) Gödel, K. (1931) afetaram também todos os matemáticos da época que se dedicavam a problemas em aberto:

> *(...) A partir de agora, para cada enunciado ainda indemonstrado, teremos que perguntar se pode ser um caso da aplicação do Teorema da Incompletude... Toda hipótese ou conjectura importante pode ser indemonstrável a priori! A afirmação de Hilbert, "na matemática não existe ignorabimus", não se aplica mais; o chão que nós pisávamos foi retirado dos nossos pés!*. Doxiadis, A. (2001)

Os Teoremas de Gödel focalizam sistemas formais expressivos o suficiente para conter a teoria dos números e enunciam (em versão informal):

(1^{o}) Todo sistema formal é incompleto, ou seja, há sempre um enunciado que pode ser expresso no sistema, mas cuja prova, não pode ser construída no próprio sistema;

(2^{o}) O enunciado que declara a consistência de um sistema não pode ser provado no próprio sistema;

O primeiro Teorema de Gödel mostrou à comunidade matemática da época que a busca pela solução de alguns dos problemas relacionados por Hilbert poderia ser inútil. O segundo Teorema foi de encontro à formalização da matemática. Demonstrar a consistência da matemática enquanto sistema formal implicaria em circularidade: usar a matemática para provar a consistência da própria matemática.

A divulgação dos Teoremas de Gödel causou uma enorme angústia dentre a comunidade matemática. Em particular, sensibilizou o então estudante de graduação em Cambridge, Alan Turing. Em 1936, Turing publicou *"On computable numbers, with an application to the Entscheidungsproblem"* Turing, A. (1936), que introduzia a noção de *computabilidade*. Turing apresenta uma máquina hipotética através da qual ele formaliza o conceito de *computável*, e mostra que o Problema de Decisão de Hilbert não tem solução.

> *We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q1, q2, ..., qR which will be called "***m***-configurations". The machine is supplied with a "tape", (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the* **r***-th, bearing the symbol S(***r***) which is "in the machine".*

No mesmo ano, um pouco antes de Turing, Alonzo Church (Church, A. (1936)), havia chegado à mesma conclusão, de forma independente, e por um caminho diverso àquele traçado por Turing.

> *In a recent paper Alonzo Church has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.*
> Turing, A. (1936)

Assim, Turing e Church atacam a terceira questão levantada por Hilbert e põem fim a abordagem formalista da matemática. Inicia-se uma nova era, onde problemas não solucionados se confundem com problemas não solucionáveis, e não há mecanismo efetivo que permita distinguir um do outro.

3. O Primeiro Teorema da Incompletude (Gödel)

> *(...) It is reasonable therefore to make the conjecture that these axioms and rules of inference*[1] *are also sufficient to decide all mathematical questions which can be formally expressed in the given systems. In what follows it will be shown that this is not the case, but rather that in both of the cited systems, there exist relatively simple problems of the theory of ordinary hole numbers which cannot be decided upon the basis axioms.*
> Gödel, K. (1934)

3.1 O Paradoxo de Richard

> Considere que todas as propriedades de números possam ser expressas em Português, a partir de alguns conceitos básicos, como o de números inteiros, soma, produto, quociente, divisível, etc. Por exemplo, poderíamos definir a propriedade de ser par como "divisível por dois". Considere uma classificação destas expressões por ordem de tamanho, e, para expressões do mesmo tamanho, por ordem alfabética. Considere ainda uma enumeração destas expressões: a menor expressão receberá o número 1, a imediatamente a seguir receberá o número 2, e assim por diante. Diremos que um número é Richardiano se ele NÃO possui a propriedade denotada por ele. Por exemplo: suponha que 15 é o número associado à expressão "é primo". Então 15 não é Richardiano, pois 15 não é primo. Seja N o número correspondente à expressão "é Richardiano". Pergunta-se N é Richardiano?
>
> Supondo que N <u>é</u> Richardiano, concluímos, pela definição da propriedade "Richardiano", que N não possui a propriedade denotada por ele. Sendo assim, N <u>não é</u> Richardiano.
>
> Supondo que N <u>não é</u> Richardiano, concluímos, pela definição da propriedade "Richardiano", que N possui a propriedade denotada por ele. Sendo assim, N <u>é</u> Richardiano.

[1] Refere-se ao sistema apresentado em Principia Mathemática (B. Russel) e ao sistema axiomático para Teoria dos Conjuntos de Zermelo-Frankael

O Paradoxo de Richard (Richard, J. (1905)) pode ser tomado como um bom ponto de partida para abordar os Teoremas de Gödel, pois ilustra de maneira informal a estratégia formalizada por Gödel na prova de seus teoremas. Assim como o Paradoxo de Russel, o Paradoxo de Richard se apóia na auto-referência e confusão proposital em que se estabelece ao se considerar a linguagem (a matemática) e a metalinguagem (a metamatemática) no mesmo patamar, e são citados, pelo próprio Gödel, como analogias evidentes à sua estratégia (Gödel, K. (1934)). O paradoxo de Richard ilustra o artifício matemático denominado *Diagonalização.*

A confusão proposital entre matemática e metamatemática. A matemática se faz presente no paradoxo através dos números, por exemplo, 1,2, etc, e operações sobre números, por exemplo 2+3, 2×3, etc. A metamatemática consiste na formulação de propriedades a respeito da matemática, por exemplo: "*p* é a menor prova de que *n* é primo". Observe a definição de número Richardiano: *um <u>número</u> é Richardiano se ele NÃO possui a <u>propriedade denotada</u> por ele.* Nela, matemática e metamatemática aparecem lado a lado.

A auto-referência. A auto-referência aparece no paradoxo ao se tentar aplicar ao número que denota uma certa propriedade a propriedade denotada por ele: *Seja N o número correspondente à expressão "é Richardiano". Pergunta-se N é Richardiano?* Como a propriedade em questão é propositalmente negativa, a aplicação afirmativa da mesma produz a contradição. Observe que a auto-referência produz o paradoxo quando é associada à propriedade negativa.

Observe que o Paradoxo de Russel, e o Paradoxo do Barbeiro de Sevilha são construídos sobre estes mesmos elementos.

Além das duas questões destacadas acima, este paradoxo também se utiliza do artifício da **enumeração**, peça fundamental das provas de Gödel. A classificação de Richard é um tanto imprecisa e baseada em suposições vagas. Mas indica a possibilidade de se criar uma classificação precisa ... como Gödel fez!

3.2 A Estratégia de Gödel

> *The formulas of a formal system (...) are, considered from outside, finite sequences of primitive symbols (variables, logical constants, and parentheses or dots) and one can easily make completely precise wich sequences of primitive symbols are meaningful formulas and which are not. Analogously, from the formal standpoint, proofs are not but finite sequences of formulas (with certain specifiable properties). Naturally, for metamathematical considerations, it makes no difference which objects one takes as primitive symbols, and we decided to use natural numbers for that purpose. Accordingly, a formula is a finite sequence of natural numbers and a prof-figure is a finite sequence of finite sequences of natural numbers. Methamatematical concepts (assertions) thereby become concepts (assertions) about natural numbers or sequences of such, and therefore (at least partially) expressible in the symbolism of*

> *the PM[2] itself. It can be shown, in particular, that the concepts "formula", "prof-figure", "provable formula" are definable within the system PM, i.e. one can produce, for example, a formula F(v) of PM with one free variable v (of the type of a sequence of numbers) such that F(v), when intuitively interpreted, says: v is a provable formula. Now we obtain an undecidable proposition of the system PM, i.e. a proposition A for which neither A nor non-A is provable, (...) Gödel, K. (1934)*

Um sistema é consistente, se nele não é possível provar contradições. Sendo assim, na busca pela matemática segura seria necessário afastar qualquer possibilidade de provar matematicamente os paradoxos. Ou seja, o sistema consistente, em que é possível expressar paradoxos, seria naturalmente incompleto. A estratégia adotada por Gödel consistiu em

(1) Reduzir a matemática à aritmética;

A aritmética era considerada a parte segura da matemática por ser concreta e finitária. Toda proposição da matemática deveria ser escrita como uma proposição da aritmética.

(2) Enumerar as proposições matemáticas;

A classificação reduziria a números tanto as expressões matemáticas quanto as expressões metamatemáticas. Qualquer destas expressões passaria a ser encarada como um número.

(3) Utilizando a enumeração obtida reproduzir a sentença auto-referente expressa informalmente no Paradoxo de Richard.

Tomar o número referente a uma proposição negativa e aplicar a proposição denotada ao número que a denota.

(4) Finalmente, demonstrar que a sentença auto-referente não pode ser provada no sistema.

Ao supor verdadeira a questão obtida pela auto-referência, conclui-se que a mesma é falsa, e ao supô-la falsa, conclui-se que é verdadeira.

Através destes quatro passos, Gödel demonstrou a existência de uma proposição expressa no sistema, que não poderia ser provada no sistema.

3.3 Números de Gödel

Para enumerar as proposições matemáticas, passo (2) da estratégia descrita na seção 3.2, Gödel criou um sistema de numeração cumprindo com os seguintes requisitos:

- Expressões distintas devem possuir números distintos.
- O número de qualquer expressão deve ser calculado de maneira precisa, em tempo finito.

---

[2] PM se refere ao sistema apresentado em "Principia Mathematica" por Russel e Whitehead, com a adição de dois axiomas por Gödel.

- Dado um número *n* e uma expressão *e*, deve ser possível decidir-se, de maneira precisa, em tempo finito, se *n* é o número de *e*.
- A expressão correspondente a qualquer número deve ser identificada de maneira precisa, em tempo finito.

A seguir, apresentamos passos da numeração de Gödel (adaptado de Kubrusly,(2007)):

($1^{\underline{0}}$) Fixar o conjunto finito de símbolos que ocorre nas expressões matemáticas: {~, $\vee$,$\rightarrow$,$\exists$,=,0,s,(,),vírgula}. A cada elemento deste conjunto, atribuir números de 1 a 10, respeitando a ordem em que estão apresentados.

($2^{\underline{0}}$) Associar números primos maiores do que 10 às variáveis independentes. Por exemplo, associar x a 11, y a 13, z a 17, e assim por diante.

($3^{\underline{0}}$) Associar o quadrado dos números primos maiores do que 10 aos símbolos de funções. Por exemplo, associar f a 11^2, g a 13^2, h a 17^2, e assim por diante.

($4^{\underline{0}}$) Associar o cubo dos números primos maiores do que 10 aos predicados. Por exemplo, associar p a 11^2, q a 13^2, r a 17^2, e assim por diante.

Nesta numeração, o número associado à expressão "Existe um x que é o sucessor de y", formalizada como ($\exists$ x) (x = s y), é o resultado de $2^{8} \times 3^{4} \times 5^{11} \times 7^{9} \times 11^{8} \times 13^{11} \times 17^{5} \times 19^{7} \times 23^{13} \times 29^{9}$.

A expressão associada ao número 1500 é "$\vee$ ~ $\rightarrow$", que resulta da decomposição de 1500 em fatores primos: $2 \times 2 \times 3 \times 5 \times 5 \times 5$, ou seja, $2^{2} \times 3^{1} \times 5^{3}$, seguida da decodificação dos expoentes 2, 1 e 3 conforme o primeiro passo.

Alguns números naturais podem resultar em expressões que não são bem formadas, e portanto, não têm significado matemático. Outros números naturais podem resultar em seqüências de primos que não contém todos os primos (por exemplo: $100 = 2 \times 2 \times 5 \times 5$, não contém 3), e por isto não são números de Gödel. Verifique, no entanto, que esta codificação cumpre com os quatro requisitos enumerados acima.

3.4 A Sentença de Gödel

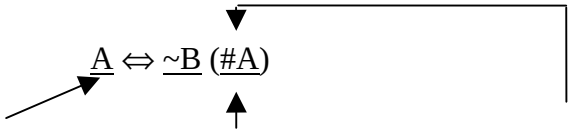
Para reproduzir a sentença auto-referente do terceiro passo da seção 3.2, considere um sistema no qual seja possível efetuar a numeração de Gödel. Vamos denotar por #p o número de Gödel da fórmula p. Definiremos p $\Rightarrow$ q como "p é a prova de q". Então, para uma dada fórmula B(x), A $\Rightarrow$ B(#A) significa que:

A é a prova de B(#A), ou seja,

A é a prova de que seu número satisfaz B.

Suponha, agora, que B expressa: "x é número de uma prova". A sentença de Gödel é uma sentença A que satisfaz: A $\Leftrightarrow$ ~B(#A): A é a prova de que seu número não é número de uma prova.

Observe a semelhança com o paradoxo de Richard.



Um número é Richardiano se ele NÃO possui a propriedade denotada por ele

Figura 3: A senteça de Gödel e o Paradoxo de Richard.

3.5 A Impossibilidade de Provar a Sentença de Gödel

Conforme descrito na seção 3.2, quarto passo, resta mostrar que a sentença auto-referente não pode ser provada. Se A é verdadeira então ~B(#A) também é. Decorre, então, que a sentença de número #A, não pode ser demonstrada, sendo portanto, falsa. Porém a sentença #A é a própria A.

Se A é falsa então ~B(#A) também é. Decorre, então, que a sentença de número #A, pode ser demonstrada, sendo portanto, verdadeira. Porém a sentença #A é a própria A.

Conclui-se então que, se A é verdadeira, então é falsa, e se A é falsa então é verdadeira, o que não é possível.

3.6 A Interpretação Computacional do Primeiro Teorema da Incompletude

O primeiro Teorema da Incompletude demonstrou para a comunidade matemática que, um sistema consistente é naturalmente incompleto, ou seja, existem sentenças que, apesar de expressas no sistema, não podem ser provadas no sistema. A prova do Teorema da Incompletude utiliza a enumeração de Gödel e do argumento diagonal e por isso só vale para sistemas em que se possam formular estes conceitos.

O computador, visto como um sistema formal, é suficientemente expressivo para suportar a enumeração de Gödel e do argumento diagonal? A resposta afirmativa a esta questão implica na validade do Teorema da Incompletude, e portanto, na existência de sentenças que, apesar de expressas no sistema, não podem ser provadas no sistema. Mas o que significa tal fato, em termos de "computador"? Deve haver uma interpretação computacional para "existem sentenças que, apesar de expressas no sistema, não podem ser provadas". Nesta interpretação computacional, o que são sentenças? O que significa "ser provada"?

O computador como um sistema formal. Na seção 1 apresentamos a Máquina de Turing (na versão apresentada em Hopcroft and Ullman (1979)) como um sistema formal. Retomamos aqui a mesma definição. Queremos mostrar a analogia deste formalismo com os computadores atuais, admitindo, porém, assim como Turing, que não há como provar a correspondência exata entre o computador e o formalismo que o pretende representar.

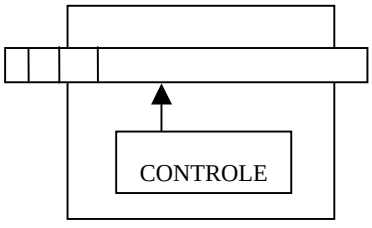


Figura 4: A Máquina de Turing

Considere que a máquina é composta por uma fita dividida em quadros. A fita é potencialmente infinita, ou seja, é finita, mas, é possível estendê-la, caso o processamento da entrada exija um maior número de quadros. Além da fita, a máquina possui um controle (cabeça ou cabeçote) que estará posicionado em um quadro a cada momento, e que efetua repetidas vezes os seguintes passos:

Lê o conteúdo do quadro em que se encontra. O conteúdo pode ser qualquer símbolo pertencente ao alfabeto da máquina, ou b (branco).

Escreve um símbolo do alfabeto ou apaga um símbolo (escreve branco) no quadro em que se encontra.

Move para o quadro imediatamente à esquerda ou à direita.

A máquina sempre inicia o processamento no estado designado como inicial, e pára quando não há instrução que possa ser efetuada.

Definição: Uma Máquina de Turing é uma 7-upla $< Q , \Gamma , b , \Sigma, \delta, Q_0, F >$ *onde,*

Q é um conjunto finito de estados;

Γ é um conjunto finito de símbolos (alfabeto da máquina);

$b \in \Gamma$ é o símbolo "branco"

$\Sigma \subseteq \Gamma - \{ b \}$ é o conjunto de entrada;

δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ é a função parcial de transição, onde L representa movimento à esquerda e R representa movimento à direita;

Q_0 é o estado inicial;

F é o conjunto de estados finais.

A função parcial de transição determina o funcionamento da máquina, já que, dados o estado corrente e um símbolo do alfabeto, ela determina o novo estado a ser alcançado, o símbolo a ser impresso e o movimento que a máquina efetuará. Por simplicidade, a função δ pode ser representada por um conjunto de quintuplas, como mostra a figura 5:

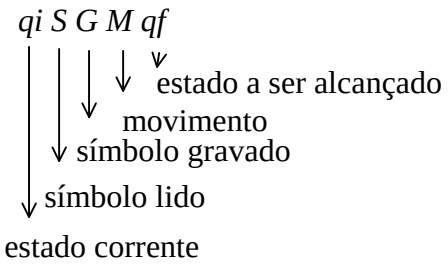


Figura 5: Representação das instruções da Máquina de Turing

O funcionamento da máquina a cada passo, descrito por uma quintupla *qi S G M qf*, pode ser lido como se segue: *"estando no estado qi, com S na fita, escreva G, mova um quadro para M (direita ou esquerda), e altere o estado para qf."*

Como δ é uma função parcial, não é preciso que todas as combinações de pares *estado corrente / símbolo lido* estejam definidas. No caso de indefinição, a máquina pára.

A Máquina de Turing como programa. Para exemplificar a Máquina de Turing, apresentamos nesta seção a Máquina de Turing como um programa que tem por finalidade efetuar a duplicação de um número lido.

Seja M a Máquina de Turing < { q_0,q_1, q_2,q_3,q_4,q_5,q_6 }, { |, 0 } , 0 , { | }, δ, q_0, {q_6} > onde δ é definido pela tabela da figura 6.

q_0	|	0	R	q_1
q_0	0	0	R	q_6
q_1	|	|	R	q_1

q_1	0	0	R	q_2
q_2	|	|	R	q_2
q_2	0	|	R	q_3

q_3	0	|	L	q_4
q_4	|	|	L	q_4
q_4	0	0	L	q_5

q_5	|	|	L	q_5
q_5	0	0	R	q_0

Figura 6: Função de Tarnsição da Máquina de Turing que efetua a duplicação da entrada

O processamento inicia com o cabeçote posicionado no primeiro caracter da entrada. A entrada da máquina, ou seja, o número a ser dobrado, consiste em uma seqüência de traços, cada traço representando uma unidade. Após haver lido um caracter da entrada, o cabeçote o apaga e avança até encontrar o primeiro branco, que indica o final da entrada. Após o branco, o cabeçote escreve dois caracteres, e retorna para duplicar o segundo caracter da entrada. Ao retornar da duplicação, o cabeçote deve retroceder os caracteres escritos na saída, o branco separador, e os caracteres que ainda restam na entrada. O processo, então se repete, com o cabeçote posicionado no primeiro caracter da entrada. No retrocesso, se não for encontrado mais nenhum caracter a ser duplicado, a máquina alcança o estado final e pára.

Na figura 7, apresentamos uma breve descrição informal da finalidade de cada estado.

q_0	|	apaga o caracter da entrada
	0	finaliza o processamento

q_1	|	avança a entrada
	0	ultrapassa o branco após a entrada

q_2	|	avança até a posição de iniciar a duplicação
	0	escreve um caracter da entrada

q_3	|	sem função definida
	0	escreve a duplicata do caracter da entrada

q_4	|	retrocede nos caracteres da saída
	0	retrocede no branco que separa a entrada da saída

q_5	|	retrocede nos caracteres da entrada
	0	reposiciona a máquina para repetir a duplicação

Figura 7: Descrição dos estados da Máquina de Turing que efetua a duplicação

A Máquina de Turing apresentada efetua uma tarefa específica, que é a duplicação da entrada. Para que ela efetue uma outra tarefa (por exemplo, a multiplicação de dois números) teríamos que considerar outra função δ, ou seja, teríamos que definir uma nova máquina $< Q,\Gamma,b,\Sigma,\delta,Q_0,F >$.

A Máquina de Turing como computador: A Máquina Universal.

> *It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with a tape on the beginning of which is written the S.D*[3] *of some computing machine $\mathcal{M}$, then $\mathcal{U}$ will compute the same sequence as $\mathcal{M}$.* Turing, A. (1936)

Para que a máquina se comporte como um computador, ela deve ser capaz de efetuar diversas tarefas, sendo cada tarefa implementada por um determinado programa. Já vimos que uma 7-upla $< Q,\Gamma,b,\Sigma,\delta,Q_0,F >$ efetua uma tarefa específica (um programa), determinada por δ. Vamos agora apresentar uma Máquina de Turing capaz de tomar um programa como entrada e efetuar o que este programa determina. Em outras palavras, a Máquina de Turing que apresentaremos aqui deverá encontrar em sua fita uma outra Máquina de Turing (o programa) e executá-lo. Chamaremos de Máquina Universal a Máquina de Turing capaz de receber programas e executá-los.

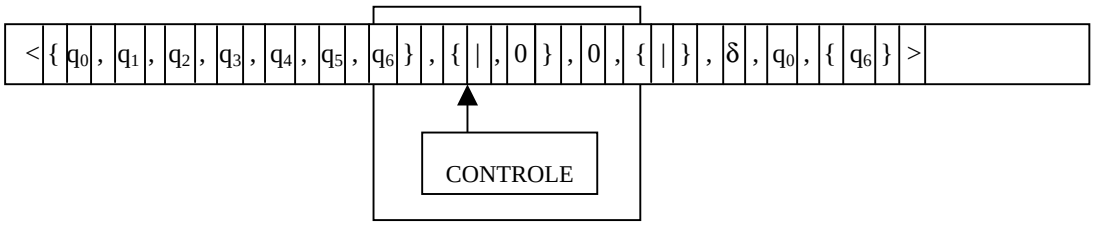


Figura 8: A máquina universal

A entrada da Máquina Universal. Para que a Máquina Universal $\mathcal{U} = < U_Q, U_\Gamma, U_b, U_\Sigma, U_\delta U_{Q0}, U_F >$ seja capaz de receber uma outra Máquina de Turing $\mathcal{M} = < Q,\Gamma,b,\Sigma,\delta,Q_0,F >$ como entrada, esta deve ser codificada no alfabeto de $\mathcal{U}$, ou seja, $\mathcal{M}$ deve ser re-escrita com os símbolos de U_Γ. Sendo U_Γ ={ |, 0 }um alfabeto binário, a quantidade de dígitos necessários para a representação de cada quíntupla da função de transição de $\mathcal{M}$ é:

> Sendo Q o conjunto de estados da $\mathcal{M}$, serão necessários x dígitos binários para codificar todos os estados, onde x é a menor potência de 2 que faz $2^x \geq |Q|$.
>
> Sendo Γ o alfabeto de $\mathcal{M}$, serão necessários y dígitos binários para codificar todos os símbolos do alfabeto, onde y é a menor potência de 2 que faz $2^y \geq |\Gamma|$.

[3] S.D. (*standard description*) é uma descrição de Máquinas de Turing usando somente os símbolos "A","C","D","R","N" e ";".

Os movimentos da máquina serão sempre *L* ou *R*, portanto, um dígito binário é suficiente para codificar os movimentos.

Desta maneira cada quíntupla da descrição do programa será representada por $2 \times x + 2 \times y^{|} + 1$, conforme mostra a figura 6.

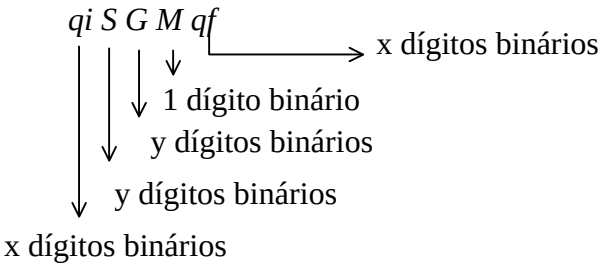


Figura 9: Tamanho da representação binária de cada quíntupla de uma Máquina de Turing

Na representação da função de transição da máquina de duplicar (figura 6) serão necessários 9 caracteres para cada quíntupla, e portanto, 99 caracteres para a representação de toda a função. Adotaremos a seguinte convenção:

Estados: $q_0 = 000$, $q_1 = 00|$, $q_2 = 0|0$, $q_3 = 0||$, $q_4 = |00$, $q_5 = |0|$, $q_6 = ||0$;

Alfabeto: 0 e | serão representados por 0 e | respectivamente;

Movimentos: L e R serão representados por 0 e | respectivamente;

A tabela da figura 10 mostra cada quíntupla da máquina de duplicar e sua respectiva codificação binária:

q_0 | 0 R q_1	000|000|
q_0 0 0 R q_6	00000|||0
q_1 | | R q_1	00||||00|
q_1 0 0 R q_2	00|00|0|0
q_2 | | R q_2	0|0|||0|0
q_2 0 | R q_3	0|00||0||
q_3 0 | L q_4	0||0|0|00
q_4 | | L q_4	|00||0|00
q_4 0 0 L q_5	|00000|0|
q_5 0 0 R q_0	|0|00|000

Figura 10: Representação binária de cada quíntupla da máquina de duplicar

A máquina de duplicar pode ser descrita no alfabeto U_T pela codificação de sua seqüência de quíntuplas:

000|000|00000|||000||||00|00|00|0|00|0|||0|00|00||0||0||0|0|00|00||0|00|00000|0||0|00|000

Observe que o processo de decodificação segue a ordem inversa dos passos da codificação. Como o tamanho de cada quíntupla, e também os tamanhos de cada componente de quíntupla são fixos, é possível saber quais os dígitos binários que correspondem a cada componente.

Além da codificação de $\mathcal{M}$, a Máquina Universal $\mathcal{U}$ deve receber também como entrada a codificação da entrada de $\mathcal{M}$ no alfabeto U_Γ . Esta codificação é feita seguindo as mesmas convenções da codificação de Γ, alfabeto de $\mathcal{M}$.

Por motivos de clareza, é conveniente considerarmos que a codificação de $\mathcal{M}$ e a codificação de seu alfabeto estão dispostas em fitas diferentes, e cada fita possui um cabeçote próprio. Na verdade, também por motivo de clareza, é conveniente considerarmos uma terceira fita auxiliar para o processo de simulação de $\mathcal{M}$. Esta terceira fita terá a função de armazenar o estado corrente de $\mathcal{M}$ na simulação. Assim, passamos a representar a Máquina de Turing como um dispositivo de três fitas, como mostra a figura 11. Neste dispositivo, cada instrução deve ser da forma $q_i\ S_1\ S_2\ S_3\ G_1\ G_2\ G_3\ M_1\ M_2\ M_3\ q_f$, onde S_k é o símbolo lido da fita k, G_k é o símbolo gravado na fita k e M_k é o movimento a ser efetuado na fita k.

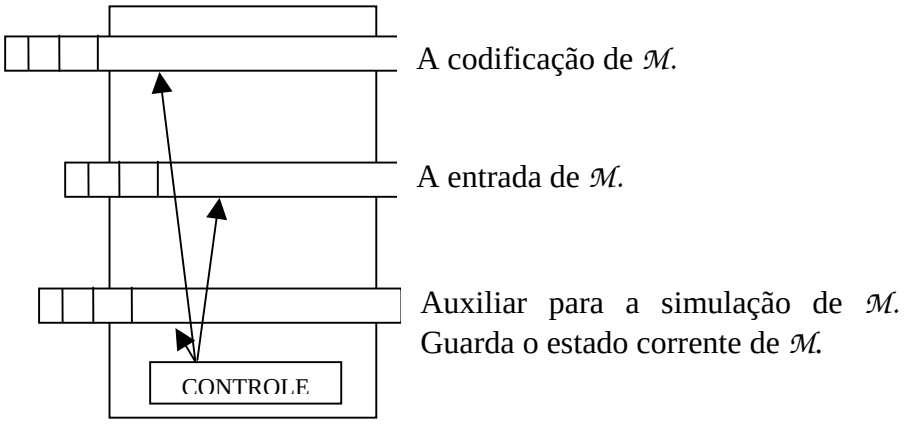


Figura 11: A Máquina de Turing de três fitas

A máquina $\mathcal{U}$, então efetua repetidamente os seguintes passos:

(1^0) Identifica o estado corrente na terceira fita.

(2^0) Identifica a entrada na segunda fita.

(3^0) Procura na primeira fita a quintupla que encabeçada pelo estado corrente, seguido do símbolo da entrada.

(4^0) Identifica a operação e o movimento na quintupla encontrada no terceiro passo.

(5^0) Efetua a operação e o movimento na segunda fita.

(6^0) Armazena o estado corrente na terceira fita.

A equivalência entre a Máquina de Turing com uma fita e a Máquina de Turing com várias fitas. Ao estendermos o modelo da Máquina de Turing de um cabeçote e uma fita para *n* cabeçotes e *n* fitas, seu poder de computação não é alterado com relação ao modelo de apenas um cabeçote e uma fita. Informalmente, este fato pode ser

percebido, já que o conteúdo das *n* fitas pode ser agrupado em *n*-uplas: as *n* primeiras quadras das *n* fitas podem ser codificadas como a primeira quadra de uma única fita, as *n* segundas quadras das *n* fitas podem ser codificadas como a segunda quadra de uma única fita, e assim por diante. Cada símbolo da nova fita deve ser associado a uma informação que indique se o cabeçote da respectiva fita está posicionado no símbolo ou não. Este esquema está representado na figura 12.

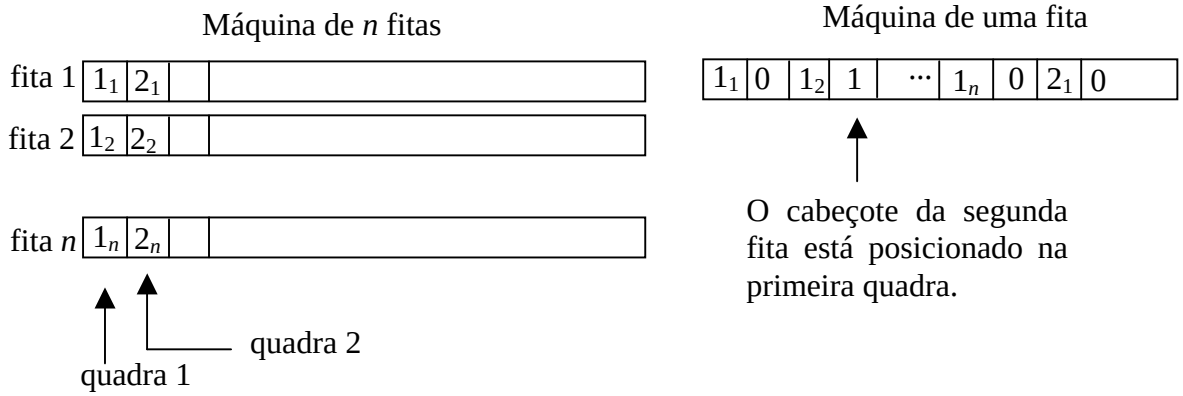


Figura 12: Simulação da Máquina de Turing de *n* fitas em uma máquina de apenas uma fita. A notação i_j significa: quadra i da fita j

Para atuar em uma fita conforme mostra a figura 12 (à direita) é necessário considerar que cada instrução da forma $q_i\ S_1 \ldots S_n\ G_1 \ldots G_n\ M_1 \ldots M_n\ q_f$ da máquina é interpretada como se segue: *"estando no estado qi, a máquina percorre a fita até encontrar as quadras consecutivas S_1 1, escreve G_1 0 no lugar de S_1 1, avança 2n + 2 símbolos no sentido M_1 da fita, e escreve 1 no lugar de 0, a seguir, percorre a fita até encontrar as quadras consecutivas S_2 1, escreve G_2 0 no lugar de S_2 1, avança 2n + 2 símbolos no sentido M_2 da fita, e escreve 1 no lugar de 0, ..., percorre a fita até encontrar as quadras consecutivas S_n 1, escreve G_n 0 no lugar de S_n 1, avança 2n + 2 símbolos no sentido M_n da fita, e escreve 1 no lugar de 0. Finalmente, altera seu estado para q_f*.

A prova formal da equivalência entre a máquina de uma fita e a máquina de *n* fitas pode ser consultada em Hopcroft, J. E. and Ullman, J. D. (1979).

Máquinas de Turing e o Primeiro Teorema da Incompletude. Retomamos agora as questões levantadas no início da seção 3.6. O computador, visto como um sistema formal, é suficientemente expressivo para suportar a enumeração de Gödel e do argumento diagonal? Há uma interpretação computacional para "existência de sentenças que, apesar de expressas no sistema, não podem ser provadas no sistema"?

Sabemos que é possível codificar uma Máquina de Turing utilizando os símbolos do alfabeto de outra Máquina de Turing. Em particular, conhecemos a codificação binária de uma Máquina de Turing. Vamos então denotar por "$\mathcal{M}$" a codificação binária da máquina $\mathcal{M}$. Observe que à qualquer seqüência binária corresponde-se um número natural. Portanto, podemos considerar uma enumeração de Máquinas de Turing: cada Máquina de Turing possui um número, não há duas máquinas com o mesmo número e não há máquina sem número.

Passamos agora ao argumento diagonal. Seguindo os passos de Gödel em sua prova, pretendemos construir um argumento auto-referente que ao ser aplicado a si próprio revela uma contradição.

Seja $\mathcal{A}$ uma Máquina de Turing que recebe como entrada um código de máquina de turing "$\mathcal{M}$"e uma entrada ω para $\mathcal{M}$ e imprime | em sua fita somente no caso em que $\mathcal{M}$, rodando com ω imprime 0. Em qualquer outro caso, $\mathcal{A}$ imprime 0:

$$\mathcal{A}(\text{“}\mathcal{M}\text{”}, \omega) = \begin{cases} | \text{ se } \mathcal{M}(\omega) = 0 \\ \\ 0 \text{ se } \mathcal{M}(\omega) \neq 0 \end{cases}$$

Seja $\mathcal{B}$ uma Máquina de Turing que recebe como entrada um código de máquina de turing "$\mathcal{M}$" aplica $\mathcal{M}$ ao seu próprio código. A máquina $\mathcal{B}$ imprime | em sua fita somente no caso em que $\mathcal{M}$, rodando com "$\mathcal{M}$" imprime 0. Em qualquer outro caso, $\mathcal{B}$ imprime 0.

$$\mathcal{B}(\text{“}\mathcal{M}\text{”}) = \mathcal{A}(\text{“}\mathcal{M}\text{”}, \text{“}\mathcal{M}\text{”}) = \begin{cases} | \text{ se } \mathcal{M}(\text{“}\mathcal{M}\text{”}) = 0 \\ \\ 0 \text{ se } \mathcal{M}(\text{“}\mathcal{M}\text{”}) \neq 0 \end{cases}$$

Questiona-se: $\mathcal{B}$, ao executar com "$\mathcal{B}$" imprime 0?

A quem responder afirmativamente, argumentamos: Pela definição de $\mathcal{B}$, $\mathcal{B}$("$\mathcal{B}$") imprimirá 0 somente no caso em que $\mathcal{B}$("$\mathcal{B}$") não imprime 0.

A quem responder negativamente, argumentamos: Pela definição de $\mathcal{B}$, $\mathcal{B}$("$\mathcal{B}$") não imprimirá 0 somente no caso em que $\mathcal{B}$("$\mathcal{B}$") imprime 0.

$$\mathcal{B}(\text{“}\mathcal{B}\text{”}) = \mathcal{A}(\text{“}\mathcal{B}\text{”}, \text{“}\mathcal{B}\text{”}) = \begin{cases} | \text{ se } \mathcal{B}(\text{“}\mathcal{B}\text{”}) = 0 \\ \\ 0 \text{ se } \mathcal{B}(\text{“}\mathcal{B}\text{”}) \neq 0 \end{cases}$$

A construção da Máquina Universal nos mostra que $\mathcal{A}$ é perfeitamente viável: para construir $\mathcal{A}$ basta utilizar a Máquina Universal para simular a execução de $\mathcal{M}$ com sua entrada, e ao final corrigir a saída para | no caso em que $\mathcal{M}$ resulta 0, e para 0 em qualquer outro caso. O motivo da contradição reside então na construção de $\mathcal{B}$. Não é possível que tal Máquina de Turing exista, apesar de seu enunciado ser extremamente claro. $\mathcal{B}$ é a sentença de Gödel, expressa em termos de Máquinas de Turing.

4. O Segundo Teorema da Incompletude (Gödel)

O Segundo Teorema afirma que a consistência de um sistema não pode ser provada dentro do próprio sistema. A estratégia de Gödel neste Teorema se utiliza do aparato lógico do Teorema anterior. Consiste em considerar uma fórmula que expresse o fato de que o sistema em questão é consistente. Através de manipulações lógicas, chega-se a conclusão que a prova da consistência implicaria na prova da sentença de Gödel, como mostra a tabela da figura 13.

1	A ⇒ ~ B(#A)	Conseqüência do Primeiro Teorema: A é a prova de que seu número é número de prova.
2	B(#(A ⇒ ~ B(#A))	Para um p arbitrário, se p é prova então #p é número de prova. (2) afirma que o número de (1) é número de prova.
3	B(#A) ⇒ B (#(~ B(#A)))	Para p, q arbitrários, se p ⇒ q é prova então #p ⇒ #q. De (1) teremos (3).
4	B(#A) ⇒ B (#(B(#A)))	Se p é prova, então o número do número da prova de p também é número de prova.
5	B(#A) ⇒ ~ CONSIST	Sabemos que um sistema é consistente se nele não é permitido provar uma sentença nem sua negação. Mas supondo B(#A) em (3), deduz-se que o número de B(#A) não é número de prova, e em (4), deduz-se que o número de B(#A) é número de prova.
6	CONSIST ⇒ ~ B(#A)	De (5) concluímos que a consistência do sistema implica no fato de que A não é número de prova.
7	~ B(#A) ⇒ A	Decorrência do Primeiro Teorema.
8	CONSIST ⇒ A	de (6) e (7). Daí, a prova da consistência implica na prova de A, que, pelo Primeiro Teorema, não pode ser provada no sistema.

Figura 13: Esboço da prova do Segundo Teorema

5. Conclusão

O que é um sistema formal? Do que um sistema formal é capaz? O computador pode ser visto como um sistema formal?

Um sistema formal é constituído por símbolos e regras precisas de manipulação destes símbolos. Utilizamos um sistema formal para representar uma dada realidade, e através de conclusões obtidas através de manipulações dentro do sistema, podemos inferir certas conclusões a respeito do que está sendo representado pelo sistema. Podemos ver um computador como um sistema formal composto de símbolos 0's e 1's, que representam níveis de tensão. As regras deste sistema formal (o computador) permitem o reconhecimento do 0, o reconhecimento do 1 e a substituição de um símbolo pelo outro em uma dada seqüência de 0's e 1's. Sendo assim, tudo o que um computador é capaz de fazer pode ser representado através de seqüências finitas de 0's e 1's. Sabemos que, a cada seqüência finita de 0's e 1's corresponde um número natural. Então, existe, no máximo, uma quantidade enumerável (do tamanho do conjunto dos números naturais) de seqüências finitas de 0's e 1's. Daí, o computador é capaz de realizar uma quantidade enumerável de processamento.

Quantas são as tarefas que gostaríamos de realizar através do computador? Se nos detivermos ao cômputo de funções no conjunto dos números naturais (f: $Nat \rightarrow Nat$), podemos chegar ao número $|Nat|^{|Nat|}$, ou seja, o tamanho do conjunto dos números

naturais elevado a ele próprio. Este valor é, no entanto maior do que a quantidade (enumerável) de seqüências finitas de 0's e 1's! Em outras palavras:

Existem mais funções que gostaríamos de computar do que possíveis programas para computá-las!

ou, em termos matemáticos:

Existem mais proposições verdadeiras do que proposições prováveis!

Este resultado nos foi demonstrado pelo primeiro teorema de Gödel: *Todo sistema formal é incompleto, ou seja, há sempre um enunciado que pode ser expresso no sistema, mas cuja prova, não pode ser construída no próprio sistema.*

Na verdade, os resultados demonstrados por Gödel são tão fortes que podem ser estendidos para além da computação e da matemática. Já que os sistemas formais são normalmente utilizados para representar processos e objetos "do mundo real", conclusões obtidas através de sistemas formais são normalmente reinterpretadas em termos "do mundo real". A experimentação nos ajuda a validar resultados obtidos formalmente, e, através destas interações entre "mundo real" e sistema formal, novos fatos são comprovados e teorias são ampliadas. No entanto, sabemos, pelo Segundo Teorema da Incompletude de Gödel, que: *O enunciado que declara a consistência de um sistema não pode ser provado no próprio sistema.* Em outras palavras:

A matemática (um sistema formal) é incapaz de provar a sua própria consistência!

Assim segundo teorema de Gödel nos faz refletir construções científicas e suas formalizações.

Por fim, o fato destes resultados terem surgido antes da invenção do computador nos leva a questionar:

É possível compreender a evolução tecnológica abordando apenas fatos e artefatos tecnológicos?

Referências

Bittencourt, G. (2006) "Computação e Computador", Universidade Federal de Santa Catarina, http://www.das.ufsc.br/gia/computer/.

Carnielli, W., Epstein, R. (2000) "Computability, Computable Functions, Logic, and the Foundations of Mathematics". Wadsworth Ed.

Church, A. (1936) "An unsolvable problem of elementary number theory", American J of Math., 58(1936), 345 – 363.

Doxiadis, A. (2001) *Tio Petrus e a Conjectura de Goldbach*. Editora Faber and Faber.

Gödel, K. (1931) "Uber formal unentscheidbare Satze der Principia Mathernatica und verwant der Systeme, I", Monatshefte Math. Phys., 38 (1931). 173-198.

Gödel, K. (1934) "On Formally Undecidable Propositions of Principia Mathematica and Related Systems, I", In: The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions. Davis, M. Lippincott Williams & Wilkins.

Hilbert, D. (1928) “Probleme der Grundlegung der Mathematik”, International Congresses of Mathematicians, Bolonha.

Hopcroft, J. E. and Ullman, J. D. (1979) Introduction to Automata Theory, Languages, and Computation, Addison-Wesley.

Kubrusly, R. (2007) “Uma Viagem Informal ao Teorema de Gödel ou (O preço da matemática é o eterno matemático)”, http://www.dmm.im.ufrj.br/projeto/diversos/godel.html

Richard, J. (1905) Les principes des mathématiques et le problème des ensembles, Revue générale des sciences pures et appliquées 16 541-543.

Turing, A. (1936), “On computable numbers, with an application to the Entscheidungsproblem”, Proceedings of the London Mathematical Society, Series 2, 42, pp 230-265.