Como estar certo mesmo estando errado

Amanda Figur

ICMC - USP

sexta-feira, 13 de novembro de 2015

Um teorema

Um teorema

O Teorema da Incompletude de Gödel

Toda teoria rica o suficiente, axiomatizável e completa é inconsistente.

Um teorema

O Teorema da Incompletude de Gödel

Toda teoria rica o suficiente, axiomatizável e completa é inconsistente.

- **O que é uma teoria axiomatizável?** É a existência de um programa executado em tempo finito que verifica se uma sequência de símbolos é uma demonstração.

Um teorema

O Teorema da Incompletude de Gödel

Toda teoria rica o suficiente, axiomatizável e completa é inconsistente.

- **O que é uma teoria axiomatizável?** É a existência de um programa executado em tempo finito que verifica se uma sequência de símbolos é uma demonstração.
- **Quando uma teoria é completa?** Se para toda afirmação existe uma demonstração para ela ou para sua negação.

Um teorema

O Teorema da Incompletude de Gödel

Toda teoria rica o suficiente, axiomatizável e completa é inconsistente.

- **O que é uma teoria axiomatizável?** É a existência de um programa executado em tempo finito que verifica se uma sequência de símbolos é uma demonstração.
- **Quando uma teoria é completa?** Se para toda afirmação existe uma demonstração para ela ou para sua negação.
- **Quando ela é inconsistente?** Quando uma afirmação e sua negação podem ser ambas demonstradas.

Um pouco de história

Provado no início de 1930 e publicado no *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*,

Um pouco de história

Provado no início de 1930 e publicado no *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*, o teorema acabou com um dos maiores sonhos de Hilbert.

Um pouco de história

Provado no início de 1930 e publicado no *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*, o teorema acabou com um dos maiores sonhos de Hilbert.

O Segundo Problema de Hilbert

Demonstrar a consistência dos axiomas da aritmética

Um pouco de história

Provado no início de 1930 e publicado no *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*, o teorema acabou com um dos maiores sonhos de Hilbert.

O Segundo Problema de Hilbert

Demonstrar a consistência dos axiomas da aritmética

Um pouco de história

Provado no início de 1930 e publicado no *Über formal unentscheidbare Sätze der Principia Mathematica und* ema acabou com um dos maiores sonhos de Hilbert.

O Segundo Problema

Demonstrar a consistênc da aritmética

Os ingredientes

Conhecimentos básicos sobre:

Os ingredientes

Conhecimentos básicos sobre:

- ~~açúcar~~ Números naturais

Os ingredientes

Conhecimentos básicos sobre:

- ~~açúcar~~ Números naturais
- ~~tempero~~ Funções

Os ingredientes

Conhecimentos básicos sobre:

- ~~açúcar~~ Números naturais
- ~~tempero~~ Funções
- ~~tudo de bom~~ Uma linguagem de programação (C, Fortran, ~~Java~~, Pascal, etc.)

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito
- A quantidade de programas possíveis é enumerável (e infinita).

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito
- A quantidade de programas possíveis é enumerável (e infinita).

Um conjunto é dito enumerável quando há entre ele e os naturais uma injeção.

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito
- A quantidade de programas possíveis é enumerável (e infinita).

Vamos dar para cada símbolo da nossa linguagem um número.

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito
- A quantidade de programas possíveis é enumerável (e infinita).

Vamos dar para cada símbolo da nossa linguagem um número. Poderíamos associar então para cada programa o número gerado pela concatenação dos números de cada símbolo.

Uma linguagem de programação

Vamos fixar uma linguagem de programação, chamamos ela de P:

- A linguagem é finita (finitos símbolos)
- Cada programa gerado é finito
- A quantidade de programas possíveis é enumerável (e infinita).

Vamos dar para cada símbolo da nossa linguagem um número. Poderíamos associar então para cada programa o número gerado pela concatenação dos números de cada símbolo.

Mas como garantir que esse número seja único?

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	$!$	$)$	$($	$=$	$+$	$.$	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	$!$	$)$	$($	$=$	$+$	$.$	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Associamos para cada um símbolo um número de três dígitos.

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	!	)	(	$=$	$+$	.	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Associamos para cada um símbolo um número de três dígitos.

A 1

B 11

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	!	)	(	$=$	$+$	.	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Associamos para cada um símbolo um número de três dígitos.

A
1

B
11

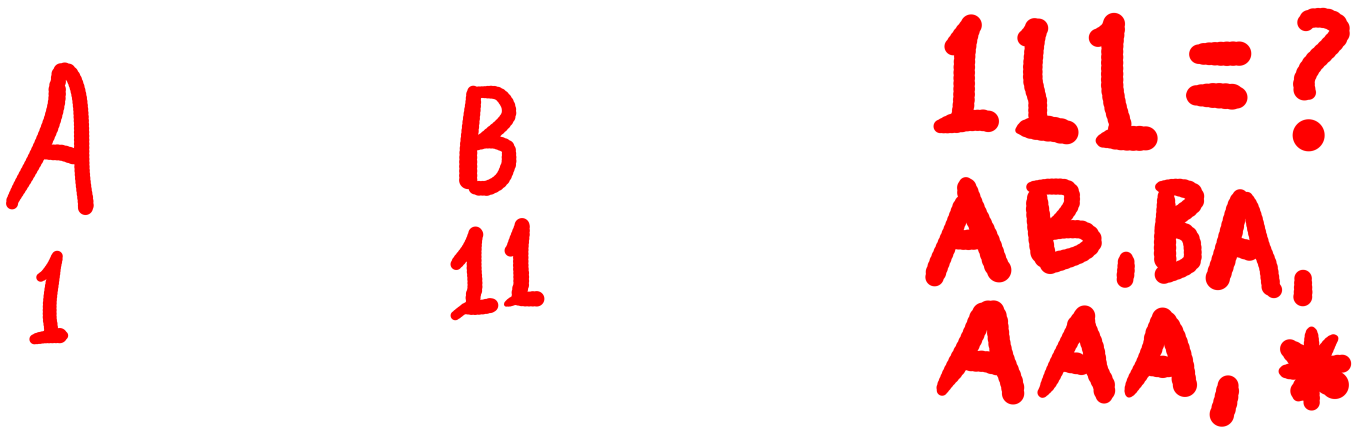
111 = ?
AB, BA,
AAA, *

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	!	)	(	$=$	$+$	.	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Associamos para cada um símbolo um número de três dígitos.



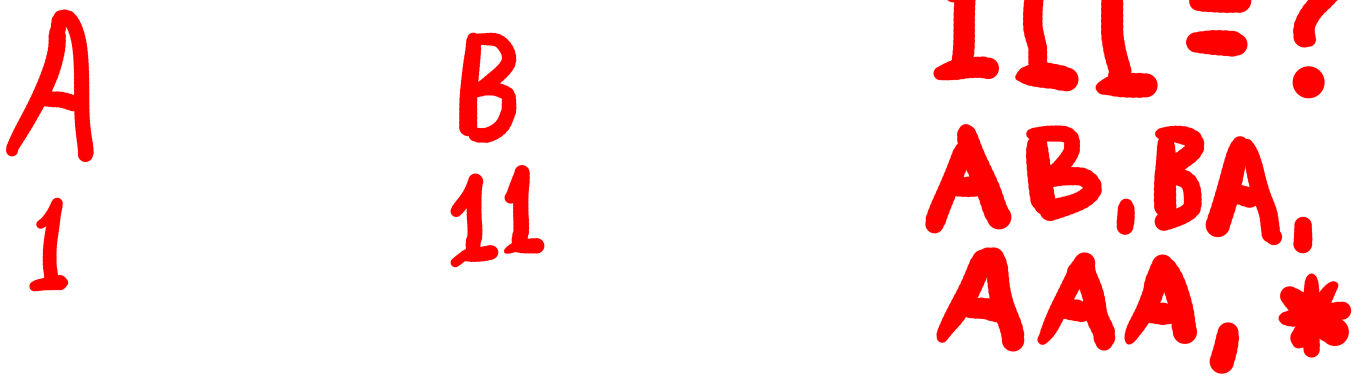
Associando um número de três dígitos para cada símbolo,

Contando os programas

Veja essa sequência de símbolos:

$\wedge$	$'$	0	$\sim$	$\neg$	!	)	(	$=$	$+$	.	$*$
100	101	102	103	104	105	106	107	108	109	110	111

Associamos para cada um símbolo um número de três dígitos.



Associando um número de três dígitos para cada símbolo, a concatenação dos números gera um número maior único.

Uma função não-computável

Definição

Seja $f : \mathbb{N} \to \{0, 1\}$. *Vamos dizer que* f *é uma* **função computável** *se existe um programa feito na linguagem* P *que, ao receber o valor* n, *devolve* $f(n)$.

Uma função não-computável

Definição

Seja $f : \mathbb{N} \to \{0, 1\}$. *Vamos dizer que* f *é uma* **função computável** *se existe um programa feito na linguagem* P *que, ao receber o valor* n, *devolve* $f(n)$.

Da definição sabemos então o que é uma função não computável:

Uma função não-computável

Definição

Seja $f : \mathbb{N} \to \{0, 1\}$. Vamos dizer que f é uma **função computável** *se existe um programa feito na linguagem P que, ao receber o valor n, devolve $f(n)$.*

Da definição sabemos então o que é uma função não computável:

- Uma função para a qual não existe programa feito na linguagem P que ao receber o valor n, devolve $f(n)$.

Uma função não-computável

Definição

Seja $f : \mathbb{N} \to \{0, 1\}$. Vamos dizer que f é uma **função computável** *se existe um programa feito na linguagem P que, ao receber o valor n, devolve $f(n)$.*

Da definição sabemos então o que é uma função não computável:

- Uma função para a qual não existe programa feito na linguagem P que ao receber o valor n, devolve $f(n)$.

Para provar que existem tais funções vamos olhar a próxima ilustração.

Enumerando as funções computáveis

$$
\begin{array}{rcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & 0, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, 0, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, 1, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, 1, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, 1, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, 0, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, 1, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, 1, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, 1, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, 1, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, 1, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, 1, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, \underline{\mathbf{1}}, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, 1, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, 1, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}, \underline{0}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, \underline{\mathbf{1}}, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, \underline{\mathbf{1}}, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, 1, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}, \underline{0}, \underline{0}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, \underline{\mathbf{1}}, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, \underline{\mathbf{1}}, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, \underline{\mathbf{1}}, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, 0, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}, \underline{0}, \underline{0}, \underline{0}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, \underline{\mathbf{1}}, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, \underline{\mathbf{1}}, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, \underline{\mathbf{1}}, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, \underline{\mathbf{0}}, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, 1, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}, \underline{0}, \underline{0}, \underline{0}, \underline{1}
\end{array}
$$

Enumerando as funções computáveis

$$
\begin{array}{lcl}
 & & 0, 1, 2, 3, 4, 5, 6, \ldots \\
f_0 & = & \underline{\mathbf{0}}, 1, 1, 0, 1, 0, 0, \ldots \\
f_1 & = & 0, \underline{\mathbf{0}}, 1, 0, 0, 1, 1, \ldots \\
f_2 & = & 1, 1, \underline{\mathbf{1}}, 0, 1, 1, 0, \ldots \\
f_3 & = & 1, 0, 0, \underline{\mathbf{1}}, 0, 0, 0, \ldots \\
f_4 & = & 0, 0, 1, 1, \underline{\mathbf{1}}, 0, 1, \ldots \\
f_5 & = & 1, 1, 0, 0, 1, \underline{\mathbf{0}}, 0, \ldots \\
f_6 & = & 0, 1, 0, 0, 1, 0, \underline{\mathbf{1}}, \ldots \\
\vdots & \vdots & \vdots \\
\hline
f & = & \underline{1}, \underline{1}, \underline{0}, \underline{0}, \underline{0}, \underline{1}, \underline{0}, \ldots
\end{array}
$$

Fixando uma função não-computável

Note que esse novo elemento, definido de tal forma, é uma função não computável.

Fixando uma função não-computável

Note que esse novo elemento, definido de tal forma, é uma função não computável.

Vamos então **fixar** $f : \mathbb{N} \to \{0, 1\}$**, tal função não computável**.

Fixando uma função não-computável

Note que esse novo elemento, definido de tal forma, é uma função não computável.

Vamos então **fixar** $f : \mathbb{N} \to \{0, 1\}$**, tal função não computável**.

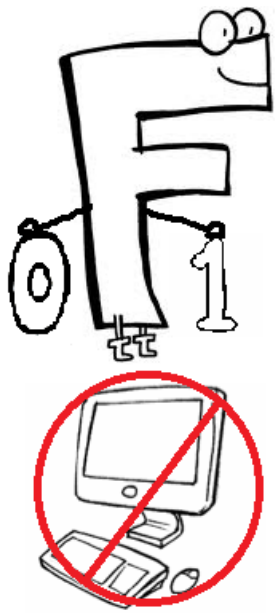
Note que para podermos explicitar a função f, basta determinar um método de ordenação para nossas funções f_n. Estando elas ordenadas, saberemos o valor de $f(n)$ pois sabemos o valor de $f_n(k)$

Fixando uma função não-computável

Note que esse novo elemento, definido de tal forma, é uma função não computável.

Vamos então **fixar** $f : \mathbb{N} \to \{0, 1\}$**, tal função não computável**.

Note que para podermos explicitar a função f, basta determinar um método de ordenação para nossas funções f_n. Estando elas ordenadas, saberemos o valor de $f(n)$ pois sabemos o valor de $f_n(k)$



Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B"

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*
- Se temos "$\varphi(x)$" e temos que "$x = y$", então temos "$\varphi(y)$". Onde φ é uma propriedade qualquer.

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*
- Se temos "$\varphi(x)$" e temos que "$x = y$", então temos "$\varphi(y)$". Onde φ é uma propriedade qualquer.

Ao fixar uma teoria T,

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*
- Se temos "$\varphi(x)$" e temos que "$x = y$", então temos "$\varphi(y)$". Onde φ é uma propriedade qualquer.

Ao fixar uma teoria T,que deve ser "rica o suficiente" para poder definir f

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*
- Se temos "$\varphi(x)$" e temos que "$x = y$", então temos "$\varphi(y)$". Onde φ é uma propriedade qualquer.

Ao fixar uma teoria T, que deve ser "rica o suficiente" para poder definir f (por exemplo, ZFC),

Uma Teoria

Definição

Vamos chamar de uma **teoria** *uma coleção de axiomas e algumas regras de inferência.*

Uma regra de inferência é uma maneira de se "obter" afirmações a partir de anteriores. **Por exemplo:**

- A partir das afirmações "A" e "$A \to B$", obtemos a afirmação "B" *(modus ponens)*
- Se temos "$\varphi(x)$" e temos que "$x = y$", então temos "$\varphi(y)$". Onde φ é uma propriedade qualquer.

Ao fixar uma teoria T, que deve ser "rica o suficiente" para poder definir f (por exemplo, ZFC), também estamos fixando os símbolos que podem ser usados nas afirmações (em geral, são coisas como $\to$, $\forall$, v, etc.).

Uma Demonstração

Definição

Fixada uma teoria, uma **demonstração** *nada mais é que uma sequência* $\varphi_1, ..., \varphi_n$ *onde cada* φ_k *é um axioma ou existem* φ_i*'s com* $i < k$ *de forma que* φ_k *é obtida a partir de uma das regras de inferência da teoria a partir das* φ_i*'s.*

Uma breve pausa

Agora vamos começar a programar.

Uma breve pausa

Agora vamos começar a programar.

É crucial que os programas possam ser executados em tempo finito...

Uma breve pausa

Agora vamos começar a programar.

É crucial que os programas possam ser executados em tempo finito...

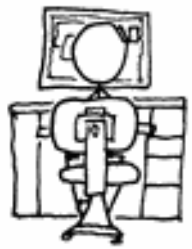
... e antes disso, um aviso:

Uma breve pausa

Agora vamos começar a programar.

É crucial que os programas possam ser executados em tempo finito...

... e antes disso, um aviso:



Programando

Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.

Programando

Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.
Estabeleçamos ele de maneira que receba um $n \in \mathbb{N}$ e devolva uma sequência de símbolos presentes em T.

Programando

Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.
Estabeleçamos ele de maneira que receba um $n \in \mathbb{N}$ e devolva uma sequência de símbolos presentes em T. Assim para qualquer demonstração existe n tal que demonstração seja `Provas` aplicado a n.

Programando

Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.
Estabeleçamos ele de maneira que receba um $n \in \mathbb{N}$ e devolva uma sequência de símbolos presentes em T. Assim para qualquer demonstração existe n tal que demonstração seja `Provas` aplicado a n.
Tal programa `Provas` pode ser definido da maneira análoga à listagem de todos os programas possíveis que exemplificamos lá em cima.

Programando

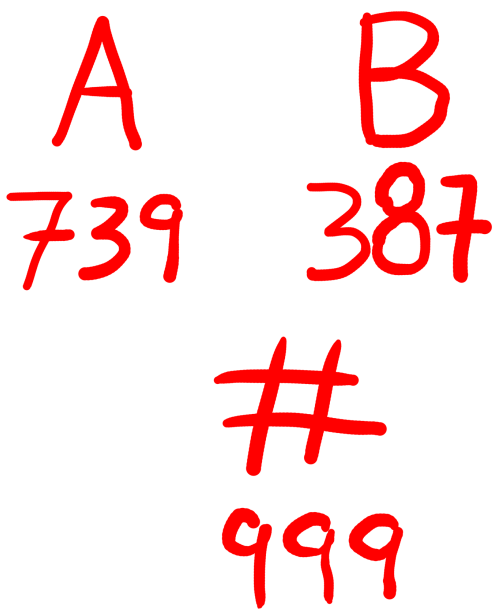
Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.
Estabeleçamos ele de maneira que receba um $n \in \mathbb{N}$ e devolva uma sequência de símbolos presentes em T. Assim para qualquer demonstração existe n tal que demonstração seja `Provas` aplicado a n.
Tal programa `Provas` pode ser definido da maneira análoga à listagem de todos os programas possíveis que exemplificamos lá em cima.

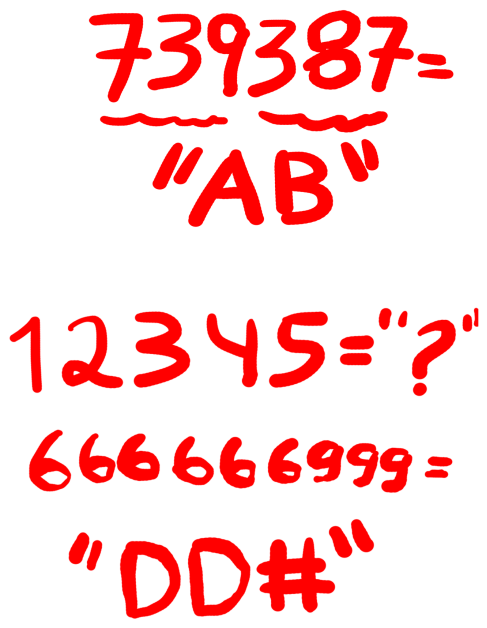
Suponha que nossa linguagem (de T) possui menos que 900 símbolos

Programando

Façamos um programa `Provas` na linguagem de programação P o qual gere todas as possíveis demonstrações da teoria T.
Estabeleçamos ele de maneira que receba um $n \in \mathbb{N}$ e devolva uma sequência de símbolos presentes em T. Assim para qualquer demonstração existe n tal que demonstração seja `Provas` aplicado a n.
Tal programa `Provas` pode ser definido da maneira análoga à listagem de todos os programas possíveis que exemplificamos lá em cima.

Suponha que nossa linguagem (de T) possui menos que 900 símbolos





Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`.

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

`EhUmaDemonstracao` recebe dois parâmetros: uma sequência de símbolos e uma afirmação φ de T.

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

`EhUmaDemonstracao` recebe dois parâmetros: uma sequência de símbolos e uma afirmação φ de T. Daí ele deve

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

`EhUmaDemonstracao` recebe dois parâmetros: uma sequência de símbolos e uma afirmação φ de T. Daí ele deve

- retornar 1 se a sequência é uma demonstração para φ;

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

`EhUmaDemonstracao` recebe dois parâmetros: uma sequência de símbolos e uma afirmação φ de T. Daí ele deve

- retornar 1 se a sequência é uma demonstração para φ;
- retornar 0 caso não seja.

Programando

Vamos implementar outro programa chamado `EhUmaDemonstracao`. Esse é o programa que atesta que nossa teoria é axiomatizável.

`EhUmaDemonstracao` recebe dois parâmetros: uma sequência de símbolos e uma afirmação φ de T. Daí ele deve

- retornar 1 se a sequência é uma demonstração para φ;
- retornar 0 caso não seja.

Uma demonstração para φ nada mais é que uma demonstração cuja última afirmação é a própria φ.

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Suponhamos T completa. Existe um programa `Provou` que recebe uma afirmação φ de T, daí ele deve

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Suponhamos T completa. Existe um programa `Provou` que recebe uma afirmação φ de T, daí ele deve

- retornar 1 se φ admite uma demonstração;

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Suponhamos T completa. Existe um programa `Provou` que recebe uma afirmação φ de T, daí ele deve

- retornar 1 se φ admite uma demonstração;
- retornar 0 se $\neg\varphi$ admite uma demonstração.

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Suponhamos T completa. Existe um programa `Provou` que recebe uma afirmação φ de T, daí ele deve

- retornar 1 se φ admite uma demonstração;
- retornar 0 se $\neg\varphi$ admite uma demonstração.

Podemos explicitar tal programa:

```
n=0;
FAZER
    SE(EhUmaDemonstracao(Provas(n), φ) == 1) RETORNA 1;
    SE(EhUmaDemonstracao(Provas(n), ¬φ) == 1) RETORNA 0;
    n++;
ENQUANTO(1=1);
```

Programando de Verdade

Definição

Dizemos que uma teoria é **completa** *se, para toda afirmação φ existe uma demonstração para φ ou uma para $\neg\varphi$ (negação de φ).*

Suponhamos T completa. Existe um programa `Provou` que recebe uma afirmação φ de T, daí ele deve

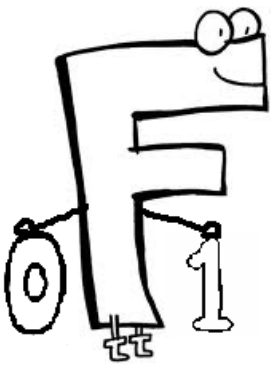
- retornar 1 se φ admite uma demonstração;
- retornar 0 se $\neg\varphi$ admite uma demonstração.

Podemos explicitar tal programa:

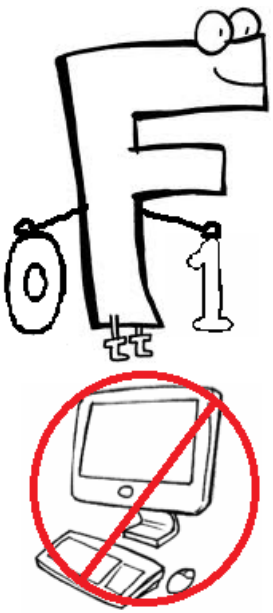
```
n=0;
FAZER
    SE(EhUmaDemonstracao(Provas(n), φ) == 1) RETORNA 1;
    SE(EhUmaDemonstracao(Provas(n), ¬φ) == 1) RETORNA 0;
    n++;
ENQUANTO(1=1);
```

Note que o programa só roda em tempo finito porque sabemos que T é completa.

De volta à função f

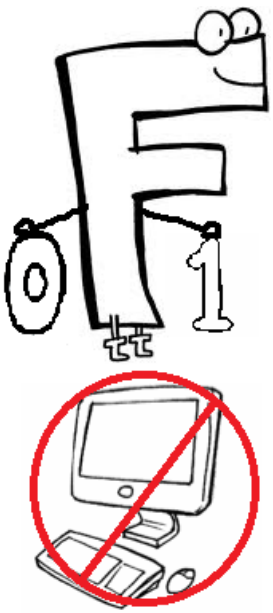


De volta à função f



Notamos que se T é completa, para cada afirmação do tipo "$f(n) = 1$" ou "$f(n) = 0$", existe uma demonstração para "$f(n) = 1$" ou uma para "$f(n) = 0$".

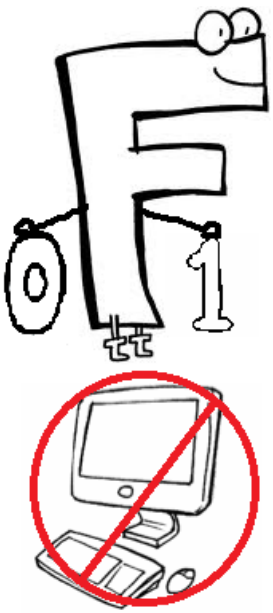
De volta à função f



Notamos que se T é completa, para cada afirmação do tipo "$f(n) = 1$" ou "$f(n) = 0$", existe uma demonstração para "$f(n) = 1$" ou uma para "$f(n) = 0$".

Como nossa teoria é completa, se não existe uma demonstração para "$f(n) = 1$", existe uma demonstração para "$f(n) \neq 1$", que é justamente "$f(n) = 0$", pois o contra domínio da nossa função possui somente esses dois elementos, 0 e 1.

De volta à função f



Notamos que se T é completa, para cada afirmação do tipo "$f(n) = 1$" ou "$f(n) = 0$", existe uma demonstração para "$f(n) = 1$" ou uma para "$f(n) = 0$".

Como nossa teoria é completa, se não existe uma demonstração para "$f(n) = 1$", existe uma demonstração para "$f(n) \neq 1$", que é justamente "$f(n) = 0$", pois o contra domínio da nossa função possui somente esses dois elementos, 0 e 1.

Definição

Dizemos que uma teoria é **consistente** *se não existe uma afirmação φ tal que existam provas para φ e para $\neg\varphi$.*

O Programa Final

Suponhamos T completa e consistente. Notamos então que, para cada afirmação do tipo "$f(n) = 1$", ou existe uma demonstração para ela,

O Programa Final

Suponhamos T completa e consistente. Notamos então que, para cada afirmação do tipo "$f(n) = 1$", ou existe uma demonstração para ela, ou existe uma demonstração para "$f(n) = 0$",

O Programa Final

Suponhamos T completa e consistente. Notamos então que, para cada afirmação do tipo "$f(n) = 1$", ou existe uma demonstração para ela, ou existe uma demonstração para "$f(n) = 0$", mas não ambos os casos simultaneamente.

O Programa Final

Suponhamos T completa e consistente. Notamos então que, para cada afirmação do tipo "$f(n) = 1$", ou existe uma demonstração para ela, ou existe uma demonstração para "$f(n) = 0$", mas não ambos os casos simultaneamente.

Faça um programa chamado `String` que recebe n e retorna a string "$f(n) = 1$".

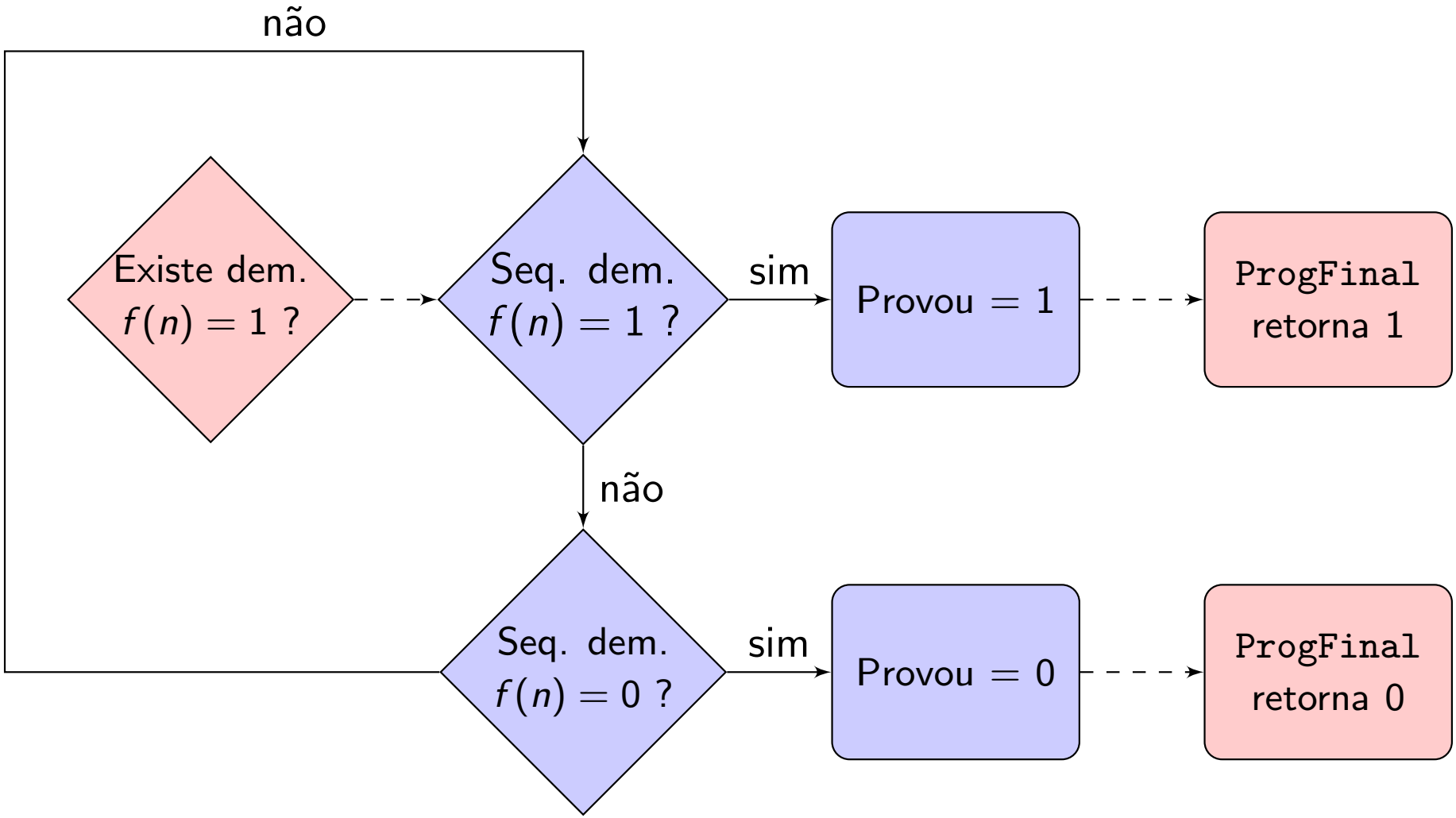
O Programa Final

Suponhamos T completa e consistente. Notamos então que, para cada afirmação do tipo "$f(n) = 1$", ou existe uma demonstração para ela, ou existe uma demonstração para "$f(n) = 0$", mas não ambos os casos simultaneamente.

Faça um programa chamado `String` que recebe n e retorna a string "$f(n) = 1$". Feito isso, basta olhar para o seguinte programa:

```
SE(Provou(String(n)) == 1)
  RETORNA 1;
SE NAO
  RETORNA 0;
```

Entendendo o Programa Final



Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável.

Entendendo o Programa Final

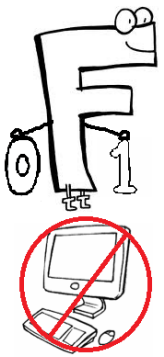
- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

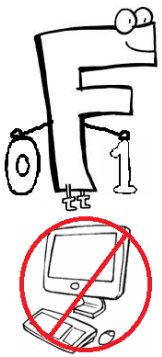
Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:



Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

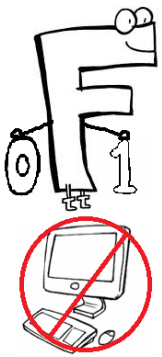
Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:



Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

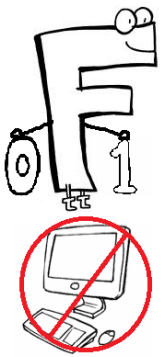
Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:



Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:

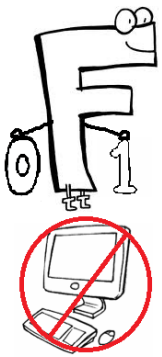


Como esse programa depende que nossa teoria seja consistente,

Entendendo o Programa Final

- Quando a afirmação "$f(n) = 1$" é demonstrável o programa retorna o valor 1;
- Quando a afirmação "$f(n) = 0$" é demonstrável o programa retorna o valor 0.

Mas isso é um programa que recebe o valor n e retorna $f(n)$, um absurdo, já que nossa f é não computável. Para entender melhor:



Como esse programa depende que nossa teoria seja consistente, provamos que uma teoria rica o suficiente para definir f e completa não pode ser consistente sem gerar contradições.

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição.

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel,

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma.

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma. Assim, T continuaria rica o suficiente para definir f,

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma. Assim, T continuaria rica o suficiente para definir f, assim como consistente.

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma. Assim, T continuaria rica o suficiente para definir f, assim como consistente. Mas, se ela for completa, então poderíamos aplicar o teorema novamente e chegar a uma nova contradição.

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma. Assim, T continuaria rica o suficiente para definir f, assim como consistente. Mas, se ela for completa, então poderíamos aplicar o teorema novamente e chegar a uma nova contradição. Concluímos que T é tal que podemos aplicar o teorema feito nessa apresentação

Considerações finais

Resumindo, assumimos T uma teoria, tal que:

- é suficientemente rica para podermos definir f;
- é consistente;
- é completa;

E chegamos a uma contradição. Desta forma, não existe uma teoria rica o suficiente para definir f, que:

- não prove contradições... (consistência)
- ...e que prove ou refute qualquer afirmação. (completude)

Como uma consequência do Teorema da Incompletude de Gödel, podemos pensar então que dada uma afirmação φ que não possui demonstração para φ nem $\neg\varphi$, bastaria acrescentar φ como um axioma. Assim, T continuaria rica o suficiente para definir f, assim como consistente. Mas, se ela for completa, então poderíamos aplicar o teorema novamente e chegar a uma nova contradição. Concluímos que T é tal que podemos aplicar o teorema feito nessa apresentação e não adianta acrescentarmos algum axioma que ela continuará sendo incompleta.

A Hipótese do Continuum

A *Hipótese do Continuum* (ou CH para os íntimos) diz que não existe X tal que

$$|\mathbb{N}| < |X| < |\mathbb{R}|$$

A Hipótese do Continuum

A *Hipótese do Continuum* (ou CH para os íntimos) diz que não existe X tal que

$$|\mathbb{N}| < |X| < |\mathbb{R}|$$

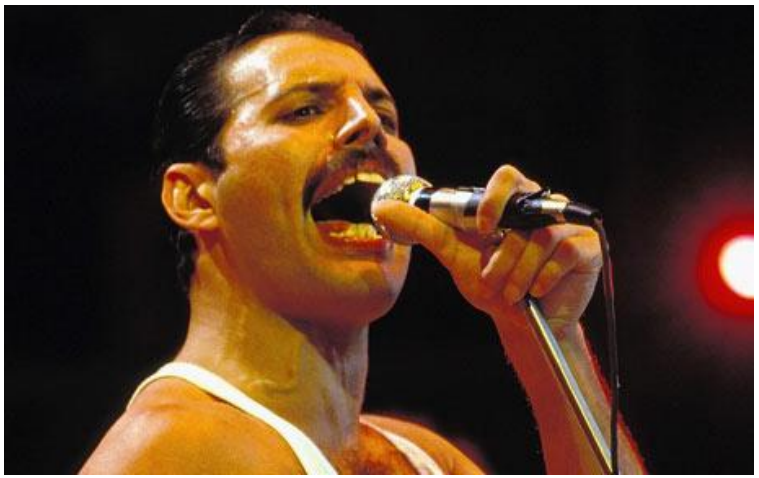
Ela foi formulada por Cantor

A Hipótese do Continuum

A *Hipótese do Continuum* (ou CH para os íntimos) diz que não existe X tal que

$$|\mathbb{N}| < |X| < |\mathbb{R}|$$

Ela foi formulada por Cantor

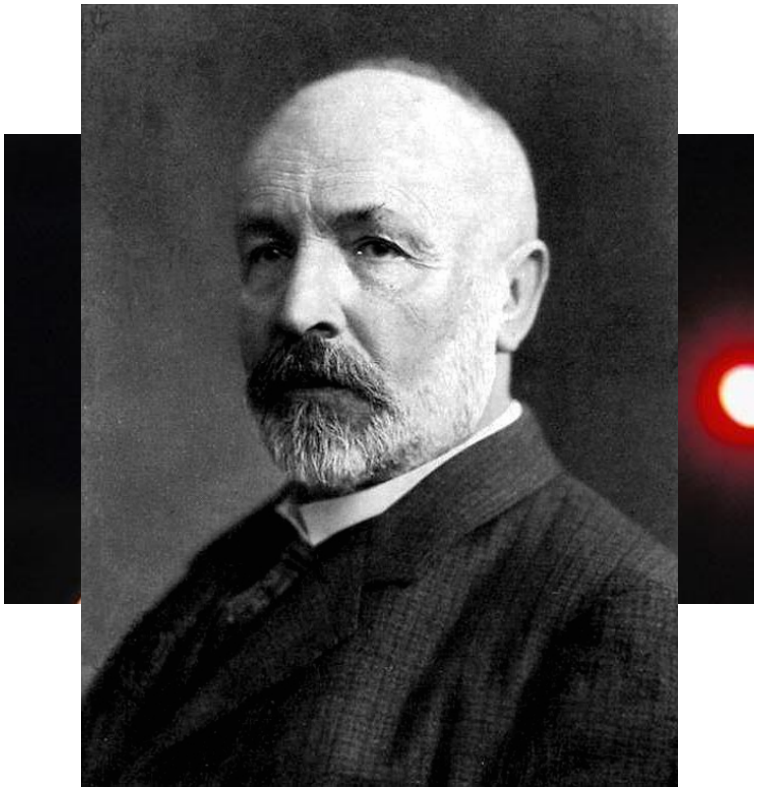


A Hipótese do Continuum

A *Hipótese do Continuum* (ou CH para os íntimos) diz que não existe X tal que

$$|\mathbb{N}| < |X| < |\mathbb{R}|$$

Ela foi formulada por Cantor



A Hipótese do Continuum

O Primeiro Problema de Hilbert

Provar a Hipótese do Continuum (CH) de Cantor.

A Hipótese do Continuum

O Primeiro Problema de Hilbert

Provar a Hipótese do Continuum (CH) de Cantor.

Gödel provou em 1940 que a CH não pode ser refutada utilizando ZFC...

A Hipótese do Continuum

O Primeiro Problema de Hilbert

Provar a Hipótese do Continuum (CH) de Cantor.

Gödel provou em 1940 que a CH não pode ser refutada utilizando ZFC...

...Paul Cohen provou em 1963 que a CH não pode ser provada também utilizando os mesmos axiomas.

A Hipótese do Continuum

O Primeiro Problema de Hilbert

Provar a Hipótese do Continuum (CH) de Cantor.

Gödel provou em 1940 que a CH não pode ser refutada utilizando ZFC...

...Paul Cohen provou em 1963 que a CH não pode ser provada também utilizando os mesmos axiomas.

A Hipótese do Continuum

O Primeiro Problema de Hilbert

Provar a Hipótese do Continuum (CH) de Cantor.

Gödel provou em 1940 que a CH não pode ser refutada utilizando ZFC...

...Paul Cohen provou em 1963 que a CH não pode ser provada também utilizando os mesmos axiomas.

Dizemos assim que a Hipótese do Continuum é independente de ZFC.

Como ganhar uma discussão

Como ganhar uma discussão

Bibliografia


Gusfield, D. (2014).
Gödel for Goldilocks: A Rigorous, Streamlined Proof of Gödel's First Incompleteness Theorem, Requiring Minimal Background.
Department of Computer Science, UC Davis.


http://www.icmc.usp.br/pessoas/aurichi/exerc/doku.php?id=lista:incompletude


http://www-history.mcs.st-andrews.ac.uk/Biographies/Godel.html


https://en.wikipedia.org/wiki/Continuum_hypothesis


https://pt.wikipedia.org/wiki/Problemas_de_Hilbert


http://www.tirinhas.com/xkcd.php?tira=292


https://xkcd.com/1081/