



Programming = Mathematics + Murphy's Law
- (E. Dijkstra)

Rafael Castro

rafaelcgs10@gmail.com

Karina Roggia

rafaelcgs10@gmail.com

07/03/2018



Sistemas de Provas

- Sistemas/Cálculos de provas servem para construir provas de uma maneira muito formal.
- São uma coleção de regras que explicam como derivar novas fórmulas.
- Um sistema de prova pode ser utilizado na formalização de diversas lógicas, como Lógica Proposicional e Lógica de Predicados.
- Os principais sistemas de provas são a Dedução Natural e o Cálculo de Hilbert.



Fragmento Proposicional Clássico da Dedução Natural

$$\frac{}{u : A} \text{ (hip)}$$

$$\frac{A \rightarrow B \quad A}{B} (\rightarrow^-)$$

$$\frac{B}{A \rightarrow B} (\rightarrow^+) [u : A]$$

$$\frac{A \wedge B}{A} (\wedge_l^-)$$

$$\frac{A \wedge B}{B} (\wedge_r^-)$$

$$\frac{A \quad B}{A \wedge B} (\wedge^+)$$

$$\frac{A \vee B \quad A \rightarrow C \quad B \rightarrow C}{C} (\vee^-)$$

$$\frac{A}{A \vee B} (\vee_l^+)$$

$$\frac{B}{A \vee B} (\vee_r^+)$$

$$\frac{\perp}{A} \text{ (efq)}$$

$$\frac{\neg \neg A}{A} \text{ (raa)}$$

onde $\neg A \Rightarrow A \rightarrow \perp$.



Exemplo de Prova em DN

$$\begin{array}{c}
 \frac{\frac{u : (A \vee \neg A) \rightarrow \perp}{\perp} \text{ (hip)} \quad \frac{\frac{v : A}{A \vee \neg A} \text{ (hip)} \quad \frac{}{A \vee \neg A} \text{ (}\vee_r^+\text{)}}{A \rightarrow \perp} \text{ (}\rightarrow^+\text{)}[v] \\
 \frac{\frac{u : (A \vee \neg A) \rightarrow \perp}{(A \vee \neg A) \rightarrow \perp} \text{ (hip)} \quad \frac{}{(A \vee \neg A)} \text{ (}\vee_l^+\text{)}}{(A \vee \neg A) \rightarrow \perp} \text{ (}\rightarrow^-\text{)} \\
 \frac{\frac{\perp}{((A \vee \neg A) \rightarrow \perp) \rightarrow \perp} \text{ (}\rightarrow^+\text{)}[u] \quad \frac{}{(A \vee \neg A)} \text{ (}\vee_l^+\text{)}}{(A \vee \neg A) \rightarrow \perp} \text{ (}\rightarrow^-\text{)} \\
 \frac{}{A \vee \neg A} \text{ (raa)}
 \end{array}$$



Interpretação BHK

A lógica intuicionista atribui um tipo diferente de semântica para fórmulas lógicas. Na lógica clássica a semântica é dada por tabela verdade, já na intuicionista é pela ideia de construção.

Pela interpretação B(L E Jan Brouwer) H(A Heyting) K(A Kolmogorov):

- uma construção de $A \wedge B$ é um par (a, b) , que indica a existência de construções a e b que resolvem, respectivamente, A e B ;
- $A \vee B$ ou é uma construção de $(0, a)$ que resolve A ou é uma construção $(1, b)$ que resolve B ;
- $A \rightarrow B$ é um procedimento tal que dadas soluções de A , obtém-se soluções de B ;
- nada é uma construção de \perp . Assim, $\neg A$ é definido como $A \rightarrow \perp$, o que significa que $\neg A$ é um procedimento que gera a constante lógica sempre falso (\perp).

A interpretação deixa livre o significado *construção*.



Cálculo Lambda

- O Cálculo Lambda é um modelo de computação criado por Alonzo Church em 1933.
- O proposito inicial do Cálculo Lambda foi ser uma linguagem de macros para um lógica e assim demonstrar a indecibilidade do problema da decisão.
- Funciona como um sistema de reescrita: existem regras para reescrever expressões.
- A primeira linguagem de programação. Uma década antes do primeiro computador.

Sintaxe:

$$e := (e \ e') \mid (\lambda x. e) \mid x$$

Reescrita:

$$(\lambda x. e) \ e' \Rightarrow_{\beta} [e'/x]e$$



Exemplos de Computação em CL

1)

$$(\lambda x.x) y \Rightarrow_{\beta} y$$

2)

$$(\lambda x.x) (\lambda x.x) \Rightarrow_{\beta} \lambda x.x$$

3)

$$((\lambda x.\lambda y.y) a)b \Rightarrow_{\beta} b$$

4)

$$(\lambda x.xx) (\lambda x.xx) \Rightarrow_{\beta} (\lambda x.xx) (\lambda x.xx)$$



Cálculo Lambda Tipado

O Cálculo Lambda é muito poderoso, permite criar o equivalente de fórmulas lógicas infinitas e assim a lógica representada é inconsistente.

Para evitar paradoxos Church utilizou o mesmo truque que Bertrand Russel: Type Thoery.



Cálculo Lambda Tipado

$$\frac{}{x : A \vdash x : A} \text{ (hip)}$$

$$\frac{\Gamma \vdash e : A \rightarrow B \quad \Gamma \vdash e' : A}{e \ e' : \Gamma \vdash B} (\rightarrow^-) \quad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x. e : A \rightarrow B} (\rightarrow^+)$$

$$\frac{\Gamma \vdash e : A * B}{\Gamma \vdash \text{fst } e : A} \text{ (fst)} \quad \frac{\Gamma \vdash e : A * B}{\Gamma \vdash \text{snd } e : B} \text{ (snd)}$$

$$\frac{\Gamma \vdash e : A \quad \Gamma \vdash e' : B}{\Gamma \vdash (e, e') : A \wedge B} \text{ (pair)}$$

$$\text{fst}(e, e') \Rightarrow_{\beta} e$$

$$\text{snd}(e, e') \Rightarrow_{\beta} e'$$



Cálculo Lambda Tipado

$$\frac{\Gamma \vdash e : A + B \quad \Gamma \vdash e' : A \rightarrow C \quad \Gamma \vdash e'' : B \rightarrow C}{\Gamma \vdash \text{case}(e, e', e'') : C} (+^-)$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash (0, e) : A + B} +_l^+ \quad \frac{B}{\Gamma \vdash (1, e) : A + B} +_r^+$$

$$\text{case}((0, e), e', e'') \Rightarrow_\beta e' e$$

$$\text{case}((1, e), e', e'') \Rightarrow_\beta e'' e$$



Isomorfismo de Curry-Howard

O Isomorfismo de Curry-Howard é uma observação que provas construtivas na dedução estão numa correspondência natural com programas em Cálculo Lambda.

$$\frac{A \rightarrow B}{B} A \quad (\rightarrow^-) \qquad \frac{B}{A \rightarrow B} (\rightarrow^+) [u : A]$$

$$\frac{\Gamma \vdash e : A \rightarrow B \quad \Gamma \vdash e' : A}{e \ e' : \Gamma \vdash B} (\rightarrow^-) \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x. e : A \rightarrow B} (\rightarrow^+)$$



Realizability

- **Realizability** atribui significado ao Isomorfismo de Curry-Howard.
- Intuitivamente: Se M é uma prova em Dedução Natural de uma fórmula A , então M resolve o problema A de acordo com a interpretação BHK.
- Essa intuição é formalizada pela **Realizability** de Kleene e Kreisel.

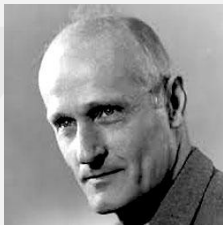


Figura: Kleene



Figura: Kreisel



O sistema de provas Minlog

- Minlog é um assistente de provas interativo: verifica a consistência de provas.
- Diferente de Coq ou Isabelle, Minlog não é baseado em *Type Theory* e sim baseado em Dedução Natural de Primeira Ordem.
- A lógica de Minlog não é a Clássica nem a Intuicionista, é a lógica Miníma.
- Minlog é o assistente de provas com o mais sofisticado sistema de extração de programas: *Realizability* e *A-Translation* permitindo extração a partir de provas clássicas.
- Além de extrair o programa, Minlog extrai a prova que o programa está de acordo com a sua especificação, assim Minlog é uma opção para *proofs-as-programs paradigm*.



Extracting a Logarithm Approximation Algorithm

$$\forall n, \exists k \, r, 0 < n \rightarrow n = 2^k + r \wedge r < 2^k$$

The intuition here is that we should increase r when $r < 2^k$ or increase k otherwise.



Sketch of the proof

Induction over n :

- ① Base case: $n = 0$. So since we have $0 < 0$, we just use efq .
- ② Step case: $n = Succ\ m$.
 - ① The base case of induction is not the base of the recursion in the algorithm, because $\log_2 0 = -\infty$.
 - ② We do case analysis on $0 < m$, so we have $0 < m$ or $m = 0$.
 - ③ This way we have two new goals on the induction step:
 - ① if $m = 0$, then $n = 1$ (our base case of recursion) and we shall prove:
$$\exists k\ r, 1 = 2^k + r \wedge r < 2^k$$

So $k = r = 0$.
 - ② if $0 < m$, then the recursion can go on and we should prove that:

$$\exists k\ r, Succ\ m = 2^k + r \wedge r < 2^k$$



Sketch of the proof

Our current goal:

$$\exists k\ r, \text{Succ } m = 2^k + r \wedge r < 2^k$$

We need to do case analysis on $\text{Succ } r < 2^k$ because that is rule for increasing k or r .

- 1 If $\text{Succ } r < 2^k$, then we chose $k = k$ and $r = \text{Succ } r'$.
- 2 If $(\text{Succ } r < 2^k) \rightarrow \perp$, then we chose $k = \text{Succ } k'$ and $r = 0$.