Como provar teoremas em um computador?

Pablo
`pablopie.xyz`

Outubro de 2022

Disclaimer

- Hacker...

```
00010011011001101011110011011101011010000101010111
10111101100000110001110010100000101100111110100000
01001110001000101000110010000001100010001100110110
00100110000010011010010100000110101110110000101100
11011111111101001010011010000111010011000111110011
```

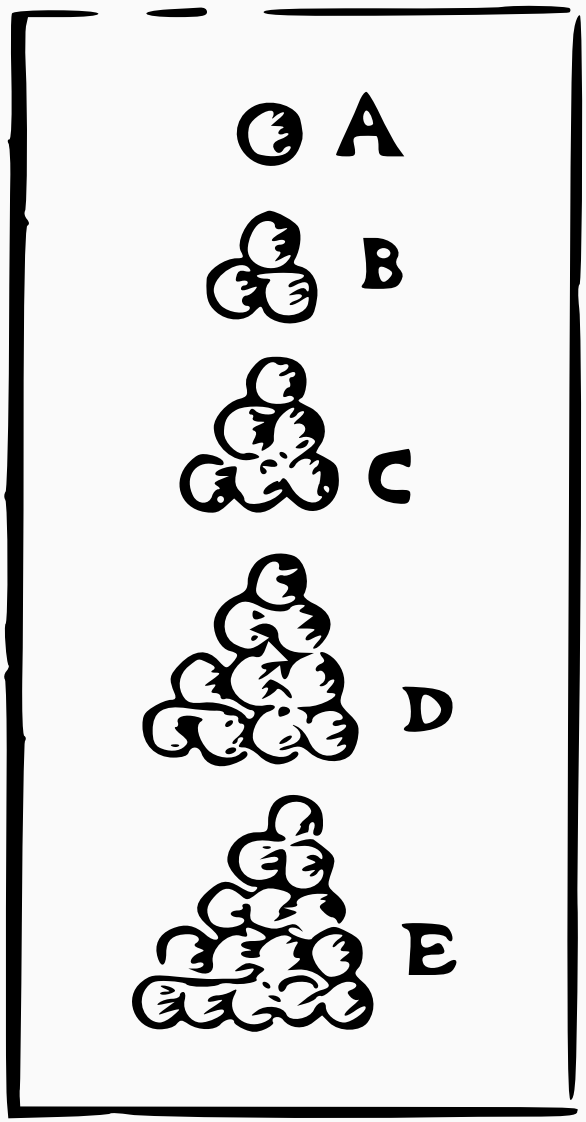
- Dedução natural...

$$\frac{\Gamma \vdash_\Sigma h_1 : P \quad \Gamma \vdash_\Sigma h_2 : Q}{\Gamma \vdash_\Sigma \mathrm{intro}\, h_1\ h_2 : P \wedge Q} \to C \qquad \frac{\Gamma \vdash_\Sigma h : P \quad \Gamma \vdash_\Sigma f : P \supset Q}{\Gamma \vdash_\Sigma f\ h : Q} \to E$$

$$\frac{\Gamma \vdash_\Sigma h : P}{\Gamma \vdash_\Sigma \mathrm{inl}\, h : P \vee Q} \to D_1 \qquad \frac{\Gamma \vdash_\Sigma h : Q}{\Gamma \vdash_\Sigma \mathrm{inr}\, h : P \vee Q} \to D_2$$

- Não precisa entender o código!
- Me interrompa!

Como provar teoremas em um computador?



- Se eu tenho um número finito de casos... posso testar todos os casos!

```
for x in X:
    assert p(x)
```

Teorema (Conjectura de Kepler)
Nenhum arranjo de esferas em três dimensões tem uma densidade média maior do que o arranjo cúbico.

- Provado em 1998 por Thomas Hales via exaustão
- E se o número de casos não for finito???

Como provar teoremas em um computador?

- Antes de tentar provar meu teorema, eu preciso enunciar ele...

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{se } n > 0 \end{cases}$$

```
def f(n):
    if n == 0:
        return 1
    else:
        return n * f(n - 1)
```

$$n! \cdot m! \mid (n+m)!$$

```
    # ???
```

- Temos termos, mas não temos formulas!

Termos	Fórmulas
Um número n	"$n > m$"
Um conjunto X	"$x \in X$"
Pontos, retas, planos, ...	O quinto postulado de Euclides

Como provar teoremas um computador?

- Objection!

$$n! \cdot m! \mid (n+m)!$$

```
f(n + m) % (f(n) * f(m)) == 0
```

$$p : \mathbb{N} \times \mathbb{N} \longrightarrow \{0, 1\}$$

$$(n, m) \longmapsto \begin{cases} 1 & \text{se } n! \cdot m! \mid (n+m)! \\ 0 & \text{caso contrário} \end{cases}$$

- Nenhum computador é capaz de checar que $p(n, m) = 1$ para todos n, m

```
for n in N:
    for m in N:
        assert p(n, m) == 1
```

A Correspondência de Curry-Howard

$$\text{Provar que vale } P \quad \leftrightsquigarrow \quad \text{Encontrar } h \in \{\text{provas de } P\}$$

- Proposições como tipos

$$\begin{aligned} P &\leftrightsquigarrow \{\text{provas de } P\} \\ P \text{ e } Q &\leftrightsquigarrow \{\text{provas de } P\} \times \{\text{provas de } Q\} \\ P \text{ ou } Q &\leftrightsquigarrow \{\text{provas de } P\} \cup \{\text{provas de } Q\} \\ P \implies Q &\leftrightsquigarrow \{\text{provas de } P\} \longrightarrow \{\text{provas de } Q\} \end{aligned}$$

- Mas como eu represento isso num computador? Tipos!

```
class AndPQ:
    """P e Q <-> AndPQ"""
    def __init__(self, h1: P, h2: Q):
        self.left = h1
        self.right = h2
```

A Correspondência de Curry-Howard

- Computadores são muito bons em checar tipos!

```
unsigned int f(unsigned int n)
{
  if (n == 0) return 1;
  else return n * f(n - 1);
}

// error: implicit conversion from 'float' to
// 'unsigned int'
f(0.5f);
```

- Linguagens estaticamente tipadas

$$
\begin{aligned}
\text{Provar que vale } P &\rightsquigarrow \text{Construir } h : \{\text{provas de } P\} \\
\text{Checar a prova} &\rightsquigarrow \text{Checar os tipos} \\
\text{Inconsistência na prova} &\rightsquigarrow \text{Tipos não batem}
\end{aligned}
$$

A Correspondência de Curry-Howard

- Mas como isso funciona na prática?

```python
class AndPQ:
    """P e Q <-> AndPQ"""
    def __init__(self, h1: P, h2: Q):
        self.left = h1
        self.right = h2
```

```
-- p e q <-> and p q
inductive and (p q : Prop) : Prop
| intro : p → q → and p q

inductive or (p q : Prop) : Prop
| inl : p → or p q
| inr : q → or p q
```

- Tipos algébricos!

A Correspondência de Curry-Howard

- Axiomas!
  - Dados x e y, existe o tipo `x = y` das provas de que $x = y$

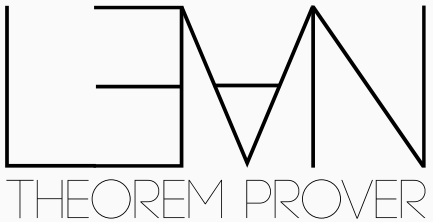
```
inductive eq (x y : t) : Prop
| rfl : eq x x

infix = := eq
```

  - O tipo `x = x` tem um único elemento `rfl`
- O tipo `x = y` varia com x e y
- Tipos dependentes!
- Codifico meus objetos como tipos

```
inductive ℕ : Type
| zero : ℕ
| succ : ℕ → ℕ
```

Proof Checkers

- Coq (`https://coq.inria.fr/`)
- Agda (`https://wiki.portal.chalmers.se/agda/pmwiki.php`)
- Lean (`https://leanprover.github.io/`)
- Idris (`https://www.idris-lang.org/`)
- Isabelle (`https://isabelle.in.tum.de/`)



Lean

```
lemma ex : or p q → or q p := by
  intro h
  match h with
  | inl h1 =>
    exact inr h1
  | inr h2 =>
    exact inl h2
```

- mathlib
  - Álgebra abstrata
  - Topologia
  - Geometria diferencial
  - Teoria dos números
  - Análise funcional
  - ...

```
variables (p' p : ℝ≥0)
  [fact (0 < p')] [fact (p' < p)] [fact (p ≤ 1)]

theorem liquid_tensor_experiment
  (S : Profinite) (V : pBanach p) :
  ∀ i > 0, Ext i (M_{p'} S) V ≅ 0 :=
```

Pra que tudo isso?

- Verificação formal de software e hardware
  - O bug do Pentium de 1994
  - A queda do voo 302 da Ethiopian Airlines em 2019
- Provas de alta complexidade
  - Classificação dos grupos simples finitos
  - A prova de Mochizuki da conjetura abc
  - A prova de Hales da conjectura de Kepler de 2015!
  - A conclusão do *Liquid Tensor Experiment* em julho de 2022!
- *Who watches the Watchmen?*

Links

1. The Natural Number Game
2. Completion of the Liquid Tensor Experiment
3. Machine-Assisted Proofs
4. Advancing mathematics by guiding human intuition with AI
5. Propositions as Types
6. Understanding typing judgments

$$\frac{\Gamma \vdash_\Sigma h : P}{\Gamma \vdash_\Sigma \mathrm{inl}\, h : P \vee Q} \to D_1 \qquad \frac{\Gamma \vdash_\Sigma h : P}{\Gamma \vdash_\Sigma \mathrm{inr}\, h : P \vee Q} \to D_2$$