Isomorfismo de Curry–Howard

Projeto submetido ao edital nº. 02/2018

Palavras-chave: lógica intuicionista, dedução natural, λ-cálculo, programação funcional.

Área de conhecimento: 1.01.01.02-0 Lógica Matemática

Resumo

Neste projeto investigamos Uma conexão interessante entre lógica e linguagens funcionais de programação.

De um modo genérico e intuitivo o isomorfismo de Curry–Howard é a afirmação de que duas famílias de formalismos aparentemente não relacionados, os sistemas formais de prova e os modelos de computação, são, de fato, o mesmo tipo de objeto matemático. Na sua forma mais elementar tal isomorfismo estabelece uma correspondência entre sentenças e suas provas em lógica minimal e valores e seus tipos em λ-cálculo simplesmente tipado. Esse último, por sua vez, fundamenta linguagens funcionais de programação de modo que temos uma correspondência entre sentenças e suas provas em um sistema lógico e valores e seus tipos em linguagem de programação. Isto permite, por exemplo, validar provas com um compilador.

1 Introdução

A Lógica como o estudo da argumentação, do raciocínio dedutivo, tem uma longa história. Os primeiros registros de um estudo sistemático são do século IV a.C., na Grécia, com o filósofo Aristóteles (384–322 a.C.). Um de seus frutos, a Lógica Matemática moderna, nasce com o trabalho do matemático britânico George Boole (1814–1864), que pela primeira vez tratou a lógica de maneira consistente como um cálculo simbólico, e também com o trabalho do filósofo alemão Gottblob Frege (1848–1925), em seu *Begriffsschrift* a lógica surge como um sistema dedutivo envolvendo axiomas e teoremas. Quando o século XIX chegou ao fim, os lógicos tinham formalizado uma noção ideal de prova.

No século XX, mais especificamente na década de 1930, uma nova geração de lógicos ingleses e americanos, incluindo Alonzo Church (1903–1995) e Alan Turing (1912–1954) dentre outros, contribuíram para a definição do conceito de algoritmo e para o desenvolvimento da teoria da computabilidade. Logo em seguida surge uma ideia de unificação das noções de prova e de computação, que vem a ser consolidada na segunda metade do século XX.

Isomorfismo de Curry–Howard é uma expressão para uma série de resultados no limite entre a lógica matemática, a ciência da computação teórica e teoria da computabilidade, que começam em 1934 com o matemático Haskell Curry (1900–1982), que notou a analogia formal entre a provas no sistema de Hilbert e a lógica combinatória, e culminam numa monografia de William Howard em 1969, mas que foi publicada somente em 1980, na qual se demonstra que as provas em dedução natural intuicionista podiam ser vistas formalmente como termos do λ-cálculo tipado (ou, tipificado). De um modo genérico e intuitivo um isomorfismo de Curry–Howard é a afirmação de que duas famílias de formalismos aparentemente não relacionados, os sistemas formais de

prova e os modelos de computação, são, de fato, o mesmo tipo de objeto matemático. Em nível sintático, no isomorfismo, fórmulas correspondem a tipos e provas correspondem a termos. Além disso, o isomorfismo fornece representações sintáticas do procedimento de construção de provas. Em particular, considerando-se uma especificação de um programa um tipo este isomorfismo permite a especificação de programas como teoremas e a obtenção de programas a partir das demonstração desses teoremas.

Esse resultado foi ponto de partida para novas direções de pesquisas levando, em particular, a uma nova classe de sistemas formais projetados para atuar tanto como um sistema de prova quanto como uma linguagem de programação funcional tipada. Esses λ-cálculos tipados derivados do paradigma Curry–Howard levaram a *softwares* como o Coq, nos quais as provas vistas como programas podem ser formalizadas, verificadas e executadas. A correspondência Curry-Howard também levantou novas questões sobre o conteúdo computacional dos conceitos de prova que não foram cobertos pelos trabalhos originais de Curry e Howard. Aplicações práticas do isomorfismo de Curry–Howard em computação estão apenas começando a serem exploradas e é uma área de pesquisa bastante ativa.

Este projeto tem por objetivo estudar o Isomorfismo de Curry–Howard entre a *Lógica Intuicionista* e o *λ-cálculo Simplesmente Tipado*. A seguir daremos um breve apresentação dos temas envolvidos neste projeto sem entrarmos nos detalhes formais e usando várias simplificações (veja [7, 6] e [8] para mais detalhes).

Lógica intuicionista. A lógica clássica, desde Aristóteles, baseia-se em três princípios fundamentais formulados a partir das necessidades práticas de filósofos e matemáticos: a lei da identidade, a lei do terceiro excluído e a lei da não contradição. Um papel importante da lei do terceiro excluído é o de fundamentar provas indiretas. Por exemplo, considere a seguinte afirmação: *existem dois números irracionais x e y tais que*

x^y *é racional.* A seguinte demonstração é um exemplo bem conhecido de uma prova indireta

> Sabemos que $\sqrt{2}$ é irracional.
>
> Considere o número $\sqrt{2}^{\sqrt{2}}$, que é racional ou é irracional.
>
> Se $\sqrt{2}^{\sqrt{2}}$ é racional, tome $x = y = \sqrt{2}$ e a afirmação está provada.
>
> Se $\sqrt{2}^{\sqrt{2}}$ é irracional, tome $x = \sqrt{2}^{\sqrt{2}}$ e $y = \sqrt{2}$ que a afirmação fica provada pois
>
> $$x^y = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^2 = 2$$
>
> que é racional.

Essa é uma prova não construtiva, ela não exibe explicitamente os números que satisfazem o enunciado, mas garante que eles existem. A afirmação "é racional ou é irracional", na segunda linha do argumento, invoca a lei do terceiro excluído. Um número real deve satisfazer a sentença "é racional" ou a sentença "não é racional", não havendo outra possibilidade.

Na lógica intuicionista a lei do terceiro excluído não é um axioma/teorema do sistema dedutivo. As provas na lógica intuicionistas são provas construtivas, ou seja, a demonstração da existência de um determinado objeto deve criá-lo ou fornecer um método para tal.

A lógica intuicionista foi desenvolvida por volta de 1930 e entre seus principais criadores estão o matemático e lógico holandês Arend Heyting (1898–1980), o alemão Gerhard Gentzen (1909–1945) e o norte-americano Stephen Kleene (1909–1994). Como sistema lógico, a lógica intuicionista tem uma linguagem bem definida e um sistema de provas consistente, munido de axiomas e regras de inferência. A sintaxe das fórmulas da lógica intuicionista é semelhante à da lógica clássica de primeira ordem, no entanto, os conectivos intuicionistas não são definíveis em termos um do outro da

mesma maneira que na lógica clássica. Na lógica intuicionista é costume usar $\rightarrow, \wedge, \vee, \bot$ como os conectivos básicos, tratando $\neg A$ como uma abreviação para $(A \rightarrow \bot)$ e ambos os quantificadores existencial e universal são necessários.

A semântica é um pouco mais complicada do que para o caso clássico. Sentenças não provadas na lógica intuicionista não recebem um valor-verdade (verdadeira ou falsa), elas permanecem de valor-verdade desconhecido até que sejam provadas ou refutadas. A ideia básica é que no caso de haver uma prova construtiva para uma proposição A ela será intuicionisticamente-verdadeira, e se houver uma prova construtiva para sua negação ela será intuicionisticamente-falsa. Uma conjunção $A \wedge B$ será intuicionisticamente-verdadeira se e somente se há provas construtivas tanto para A quanto para B, e será falsa quando tivermos uma construção para $\neg A$ ou para $\neg B$ ou ambas. Do mesmo modo, $A \vee B$ será intuicionisticamente-verdadeira se temos uma prova construtiva para A ou uma para B, ou para ambas, e falsa se tivermos provas para ambas as negações. Uma implicação $A \rightarrow B$ é intuicionisticamente-verdadeira se temos um método construtivo tal que, de uma prova construtiva para A, podemos obter uma prova construtiva para B.

A lógica intuicionista é mais fraca que a lógica clássica, cada teorema da lógica intuicionista é um teorema da lógica clássica, porém tautologias na lógica clássica não são teoremas da lógica intuicionista. A lei do terceiro excluído, como já foi dito, é um exemplo de tautologia clássica que não é teorema intuicionista e também a lei de Pierce $((A \rightarrow B) \rightarrow A) \rightarrow A$ e a eliminação de dupla negação $\neg\neg A \rightarrow A$ não são teoremas da lógica intuicionista (curiosamente, a introdução de negação dupla $A \rightarrow \neg\neg A$ é um teorema intuicionista).

λ-cálculo. O λ-cálculo é um sistema formal introduzido na década de 1930 pelo matemático norte-americano Alonzo Church como parte de um sistema para a fundamen-

tação da Matemática, entretanto, seus alunos Stephen Kleene e John Rosser provaram, em 1935, que o sistema era inconsistente. Em 1936, Church percebeu que os termos da sua linguagem poderiam ser usados para expressar as funções computáveis e, então, isolou a parte do sistema que lidava apenas com funções e o usou para estudar a computabilidade. Na mesma época, independentemente, o inglês Alan Turing escreveu o famoso artigo no qual define a máquina que leva seu nome. Não demorou muito tempo para que se provasse que as duas formulações eram equivalentes, tudo o que pode ser computado no λ-cálculo pode ser computado por uma Máquina de Turing e vice-versa.

A ideia básica do λ-cálculo é que tudo é função. Usualmente nós definimos uma função por uma equação; se uma função f é definida pela equação $f(x) = t$, onde t é algum termo envolvendo x, então a aplicação $f(u)$ produz o valor $t[u/x]$, onde $t[u/x]$ é o termo que resulta substituindo u para cada ocorrência de x em t. Por exemplo, se $f(x) = x \cdot x$, então $f(3) = 3 \cdot 3 = 9$. Church estabeleceu uma maneira especialmente compacta de escrever tais funções, ele simplesmente escreveu $\lambda x.t$. Em sua notação o nosso exemplo fica $\lambda x.x \cdot x$. A essência do cálculo é descrita pela *regra de redução*: $(\lambda x.t)(u) \to_{\mathrm{r}} t[u/x]$. Por exemplo, $(\lambda x.x \cdot x)(3) \to_{\mathrm{r}} 3 \cdot 3 \to_{\mathrm{r}} 9$.

As funções são definidas através de um λ-termo que dita o comportamento da função. Em particular, uma função é explicitada por uma expressão que pode conter funções que ainda não estão definidas e que são substituídas por variáveis. Se t é λ-termo, então $\lambda x.t$ é também um λ-termo e representa a função que associa x com t. Por exemplo, podemos adicionar o termo $\langle x, y\rangle$ para construir um par ordenado e os termos $p.pri$ e $p.seg$ para selecionar a primeira e a segunda coordenada, respectivamente, de um par ordenado. As regras de redução são $\langle x, y\rangle.pri \to_{\mathrm{r}} x$ e $\langle x, y\rangle.seg \to_{\mathrm{r}} y$. A função

que inverte as coordenadas de um par é o termo

$$\lambda z.\langle z.seg, z.pri\rangle \tag{1}$$

que aplicada a um par $\langle a, b\rangle$ fica

$$(\lambda z.\langle z.seg, z.pri\rangle)(\langle a, b\rangle) \to_{\mathrm{r}} \langle\langle a, b\rangle.seg, \langle a, b\rangle.pri\rangle \to_{\mathrm{r}} \langle b, a\rangle$$

O λ-cálculo pode ser não-tipado ou tipado. No λ-cálculo não-tipado as funções não possuem domínio e contradomínio específicos. Todo λ-termo é, ao mesmo tempo, função e argumento. Dessa maneira um λ-termo pode ser aplicado a qualquer outro λ-termo, inclusive a si mesmo, o que pode gerar inconsistência. Por exemplo, se f é dada pelo termo $\lambda x.x$ então $f(f)$ é $\lambda x.x(f) = f$, que não faz sentido na matemática ordinária. Para resolver os problemas decorrentes desta característica introduziu-se um sistema de tipos. Tipos são objetos de natureza sintática que desempenham um papel análogo ao de conjunto e os termos, por sua vez, representam o papel de elemento de um conjunto. Todo λ-termo deve ter um tipo especificado. Termos e tipos são gerados livremente a partir de um alfabeto e de operações binárias. Mais formalmente, os tipos simples são construídos da seguinte forma: um tipo básico τ (podemos dar vários tipos de bases, como inteiro, booleano e outros); se τ_1 e τ_2 são tipos, $\tau_1 \to \tau_2$ é um tipo (representa as funções que com argumentos do tipo τ_1 e retornam um elemento do tipo τ_2).

O λ-cálculo constitui a base de todas as linguagens de programação funcional. O Lisp, por exemplo, é uma linguagem funcional não-tipada e o OCaml é uma linguagem funcional tipada. Desse modo, em última instância, um prova é "o mesmo" que um programa, via isomorfismo de Curry–Howard. Por exemplo, os termos do exemplo acima, sobre trocas de coordenadas de um par, podem ser escrito em sintaxe de OCaml como

```
let par   = (fun x -> (fun y -> (x,y)))
let prim  = (fun x -> match x with (a,b) -> a)
let seg   = (fun x -> match x with (a,b) -> b)
let troca = (fun x -> (par (seg x) (prim x)))
```

e tais termos/funções têm seus correspondentes na prova em Dedução Natural

$$\dfrac{\dfrac{\dfrac{A \wedge B}{B}\,(\text{E2}) \qquad \dfrac{A \wedge B}{B}\,(\text{E1})}{B \wedge A}\,(\wedge)}{A \wedge B \to B \wedge A}\,(\to)$$

que usa as regras de inferência

$$\dfrac{A \wedge B}{A}(\text{E1}), \quad \dfrac{A \wedge B}{B}(\text{E2}), \quad \dfrac{\begin{array}{c} A \\ B \end{array}}{A \wedge B}(\wedge) \quad \text{e} \quad \dfrac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{A \to B}(\to).$$

O λ-cálculo e os sistemas formais dedutivos têm um papel fundamental em várias subáreas da computação como a programação funcional, a semântica das linguagens de programação, a demonstração automática de teoremas e o processamento de linguagem natural.

2 Objetivos

O objetivo geral desse projeto é iniciar formação científica do aluno, introduzi-lo na prática da pesquisa científica e aprimorar sua qualificação para que possa contribuir com o tema no futuro. No processo buscaremos que o aluno adquira familiaridade com conceitos abstratos formais de sistema dedutivo, de prova e de modelo de computação, entenda o contexto dos conceitos e objetivos de um sistema formal e de um

modelo abstrato de computação. Também, pretendemos que o aluno apure as habilidades matemáticas relevantes para a área e desenvolva leitura e compreensão de textos científicos.

Como um subproduto esperamos ter, no final deste projeto, um texto em português sobre o tema (até o momento não encontramos um) que sirva como referência que possa ser apresentada aos alunos da disciplina "Lógica Básica" e que traga um novo ponto de vista para que se perceba a importância da matemática na sua formação exibindo uma relação direta entre dois objetos do seu cotidiano acadêmico.

3 Metodologia e Cronograma

A metodologia é tradicional ao estudo teórico. Faremos um estudo guiado do tema, a partir de desafios propostos pelo orientador o aluno deverá buscar os conceitos, técnicas e desenvolver a solução a fim de atingir o objetivo de desenvolver no aluno as habilidades relacionadas ao desenvolvimento de pesquisa científica.

Para atingir a meta que especificamos acima dividimos o tema em quatro tópicos

1. Sintaxe da Lógica proposicional intuicionista;
2. Dedução Natural para lógica intuicionista;
3. Teoria de tipos simples para o λ-cálculo;
4. Isomorfismo de Curry-Howard;
5. Aplicações e as possíveis extensões (sistemas estilo-Hilbert, lógica proposicional clássica, lógica de primeira e segunda ordem intuicionista);
6. Redação de relatório.

Propomos o seguinte cronograma em função dos itens dispostos acima:

	10/18	11/18	12/18	01/19	02/19	03/19	04/19	05/19	06/19	07/19	08/19
1	×	×									
2		×	×	×							
3					×	×	×				
4							×	×	×	×	
5									×	×	×
6				×	×				×	×	×

As referências principais para o estudo são: [7, 2, 9]. Outras referências que usaremos são dadas abaixo.

Referências

[1] Jean Gallier. Constructive logics: Part I: A tutorial on proof systems and typed λ-calculi. *Theor. Comput. Sci.*, 110(2):249–339, March 1993.

[2] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[3] Stéphane Graham-Lengrand. The curry-howard view of classical logic a short introduction, 2016. Disponível em http://www.lix.polytechnique.fr/~lengrand/Work/Teaching/MPRI/Notes.pdf, acesso Jul/201.

[4] J Roger Hindley. *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science 42. Cambridge University Press, 1997.

[5] Martin Wirsing Iman Poernomo, John N. Crossley. *Adapting Proofs-as-Programs: The Curry-Howard Protocol*. Monographs in Computer Science. Springer, 1 edition, 2005.

[6] Jonathan P. Seldin J. Roger Hindley. *Lambda-calculus and combinators, an introduction*. Cambridge University Press, 2008.

[7] G. E. Mints. *A short introduction to intuitionistic logic*. University series in mathematics (Plenum Press). Kluwer Academic / Plenum Publishers, Springer, 2000 edition, 2000.

[8] G. E. Revesz. *Lambda-calculus, Combinators and Functional Programming*. Cambridge Tracts in Theoretical Computer. Cambridge University Press, 1988.

[9] M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2006.

[10] Jørgen Steensgaard-Madsen. Programs as proofs, 2015. arXiv:1509.04040.