1 Introdução

1.1 Isomorfismo de Curry-Howard

Esta seção visa mostrar como as provas escritas em lógica intuicionista correspondem a programas da linguagem funcional. Para tanto, utilizaremos a noção de tipos das linguagens de programação a fim de relacionar as proposições intuicionistas com estes. A relação que construiremos será a seguinte. Sejam P uma proposição qualquer e P' qualquer tipo correspondente. Deverá existir, então, uma prova para P se e somente se houver algum termo válido do tipo P'. Este termo será computado por um programa, razão pela qual designaremos como provas os programas cujo objetivo seja esta correspondência. A fins de simplificação, falaremos em existência de termos quando eles forem corretamente tipados; assim, poderemos poupar a palavra válido após qualquer frase de existência.

Inicialmente demonstremos que, para quaisquer proposições P e Q e respectivos tipos correspondentes P' e Q', existe uma prova para $P \star Q$ se e somente se há um termo válido do tipo $P' * Q'$. Aqui $\star$ é um conectivo lógico e $*$, o conjunto de todos os construtores para o tipo correspondente $P' * Q'$.

Supondo $\star = \vee$, podemos ter dois construtores para $* = \oplus$:

$$\oplus := \{\mathsf{left}, \mathsf{right}\}$$

em que $\mathsf{left} : P' \to P' \oplus Q'$ é o construtor que recebe como entrada termo do tipo P' e $\mathsf{right} : Q' \to P' \oplus Q'$ é o que recebe termo do tipo Q'. Esse isomorfismo é facilmente validado deste modo. ($\Rightarrow$) Por definição, $P \vee Q$ tem prova se (i) P ou (ii) Q têm prova. Assim, há (i) $p : P'$ ou (ii) $q : Q'$ e, portanto, (i) $\mathsf{left}(p)$ ou (ii) $\mathsf{right}(q)$. ($\Leftarrow$) Já se há $x : P' \oplus Q'$, então x é gerado por um dos construtores: (i) $\mathsf{left}(p)$ com algum $p : P'$ ou (ii) $\mathsf{right}(q)$ com algum $q : Q'$. Dessarte, há prova para (i) P ou (ii) Q, como queríamos demonstrar.

De maneira semelhante, para $\star = \wedge$ e $* = \otimes$:

$$\otimes := \{\mathsf{and}\}$$

em que $\mathsf{and} : P' \to Q' \to P' \otimes Q'$ é o construtor que recebe termos dos tipos P' e Q' respectivamente. ($\Rightarrow$) Se há prova para $P \wedge Q$, então também há para P e Q – existem, pois,

$p : P'$ e $q : Q'$. Por conseguinte, $\mathsf{and}(p, q)$ é termo válido do tipo $P' \otimes Q'$. ($\Leftarrow$) Por outro lado, quando há $x : P' \otimes Q'$, $x = \mathsf{and}(p, q)$ para algum $p : P'$ e algum $q : Q'$. Logo, deve existir prova para $P \wedge Q$.

No caso da implicação, o tipo correspondente é o funcional; ou seja, $P \Rightarrow Q$ é passível de prova se e somente se existe função do tipo $P' \to Q'$. Para entender essa correspondência, vale visualizar a estrutura de uma função $f : P' \to Q'$:

$$f(p : P') := \textbf{return}\ q : Q'$$

Toda função se constitui de um escopo, conjunto de todas as variáveis que são usadas ou não por ela para computar um resultado. Neste contexto, as variáveis[1] – que são termos corretamente tipados – correspondem a provas ou hipóteses de certas proposições. O escopo engloba as variáveis globais – que seriam os teoremas já provados e axiomas – e os argumentos. Assim, ao inserir-se um argumento, supõe-se um novo termo no escopo da função, o que equivale à noção de implicação ou suposição lógica.

Uma regra de inferência muito empregada na prova de teoremas é a eliminação da implicação ou *modus ponens*:

$$\frac{P \Rightarrow Q, P}{\therefore Q}$$

que é facilmente obtida na programação a partir da aplicação da função correspondente à prova de $P \Rightarrow Q$ sobre o termo de tipo correspondente de Q:

$$f : P' \to Q'$$

$$p : P'$$

$$f(p) : Q'$$

conforme havíamos definido.

A negação lógica $\neg P$ é definida a partir da implicação:

$$\neg P := P \Rightarrow \mathsf{F}$$

Outrossim, o tipo correspondente pode ser

$$\sim P' := P \to \mathsf{F}'$$

[1] Considerando apenas as variáveis atribuídas

sendo F' um tipo vazio, sem construtores, o que significa

$$\nexists p : \mathsf{F}'$$

Portanto, nunca haverá prova para a proposição correspondente, identicamente à proposição vazia.

Se quisermos adicionar, ainda, um tipo a corresponder com a proposição tautológica, bastará-nos definir um construtor sem argumentos; por exemplo, $\mathsf{true} : \varnothing \to \mathsf{T}'$. Claramente, $\mathsf{true}()$ serve de prova para a tautologia.

Utilizamos, muitas vezes, quantificadores lógicos em nossas proposições, como:

$$Q := \forall x : X, P(x)$$

em que P é uma proposição que depende do valor de x, sendo X um tipo qualquer. Seguindo nosso raciocínio, o correspondente de P deve ser um tipo dependente, da mesma maneira. Um termo de Q' seria da forma

$$f(x : X) := \textbf{return}\ p : (P'(x))$$

Sendo assim, f seria capaz de gerar um termo válido p do tipo que dependesse de qualquer argumento. Em outras palavras, f retornaria, para todo e qualquer $x : X$ de entrada, um termo do tipo dependente $P'(x)$ – o isomorfismo faz-se evidente. Aqui reside um possível questionamento: qual é o tipo de f? Explicitar isso visando a um entendimento fácil não é muito trivial, mas no CIC seria feito semelhantemente a

$$f : (\forall' x : X, P'(x))$$

estreitando a relação entre tal cálculo e a lógica intuicionista.

Já o quantificador existencial

$$Q := \exists x : X, P(x)$$

é demonstrado por meio de um $x : X$ qualquer e de uma prova de $P(x)$. Sejam $x_0 : X$ um termo qualquer, o construtor existencial

$$\mathsf{exist} : (\forall' x : X, P'(x) \to \exists' x : X, P'(x)))$$

e $p : P'(x_0)$ o correspondente de prova para $P(x_0)$, o termo $\mathsf{exist}(x_0)(p) : (\exists' x : X, P'(x))$ corresponde a uma prova de Q.

Uma relação importante e que estará presente no decorrer deste trabalho é a de igualdade. Possivelmente determinamos um construtor para o tipo correspondente dela destarte:

$$\mathsf{refl}(x : X) : (x =' x)$$

novamente mediante um tipo dependente. Além disso, para demonstrar diversas igualdades e outras relações, precisamos da técnica de prova por indução, que, no presente isomorfismo, corresponde à recursão de funções. De modo a simplificar a elucidação, tenhamos de exemplo o tipo dos naturais ($\mathbb{N}$). Vamos representar esses números na forma unária, já que é fácil de compreender e implementar. Para tanto, usaremos dois construtores: um vazio e outro que recebe um natural (de maneira recursiva). Aquele representará o zero (0) e este ($S(n : \mathbb{N})$), o sucessor de n. Assim, na prática da nossa representação, teremos os termos da Tabela 1.

Representação unária	**Decimal**
0	0
$S(0)$	1
$S(S(0))$	2
$S(S(S(0)))$	3
$S(S(S(S(0))))$	4
$S(S(S(S(S(0)))))$	5
$\vdots$	$\vdots$

Tabela 1 – Naturais em representação unária e decimal

Agora, examinemos a função

$$\begin{aligned} f(p_0, p_n) := \ & \mathsf{rec}(0) := p_0 \\ & \mathsf{rec}(S(n)) := p_n(n)(\mathsf{rec}(n)) \\ & \textbf{return } \mathsf{rec} \end{aligned}$$

em que $p_0 : P'(0)$ corresponde à hipótese de alguma proposição P sobre o zero e p_n é do tipo $\forall' n : \mathbb{N}, P'(n) \to P'(S(n))$. Dentro da função f, definimos outra função, rec, separando em casos distintos para cada construtor dos naturais. Observando os tipos acima, notamos que p_0 equivale à base da prova por indução e $p_n(n, h)$ para qualquer $n : \mathbb{N}$, ao passo indutivo, sendo h o equivalente à hipótese de indução.

Uma prova completa da correspondência de Curry-Howard vai além (SILVA, 2017). Não obstante, com o exposto, podemos captar as noções principais para o entendimento do funcionamento do assistente de provas Coq neste trabalho de conclusão de curso. Tal sistema computacional funciona como um verificador de provas ao passo que valida os programas que construímos (ou seja, provas).

Referências

SILVA, R. C. G. *Visão Categórica do Sistema de Tipos de Haskell*. Monografia (Trabalho de Conclusão de Curso) — Bacharelado em Ciência da Computação, Universidade do Estado de Santa Catarina, Joinville, 2017. Disponível em: <http://sistemabu.udesc.br/pergamumweb/vinculos/000043/000043b8.pdf>. Acesso em: 12 mar. 2021.