Математическая логика и логическое программирование

mk.cs.msu.ru → Лекционные курсы
→ Математическая логика и логическое программирование (3-й поток)

Вне программы 04

Изоморфизм Карри-Говарда

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2024/2025, осенний семестр

В середине XX века математиками и логиками Х.Б. Карри и В.А. Говардом постепенно была замечена, исследована и формализована схожесть устройства логических доказательств (особенно в интуиционистском натуральном исчислении) и программ в функциональной парадигме, которую принято называть изоморфизмом Карри-Говарда[1]

Фундаментом этой схожести является знак «$\to$» и запись «$A \to B$»:

- В логике эта запись означает формулу-импликацию
- В функциональном программировании (точнее, в просто типизированном λ-исчислении и всём, что на нём основано) эта запись означает тип отображений элементов типа A в элементы типа B

1 Слово «изоморфизм» здесь используется вольно, нестрого и совершенно не так, как, например, для графов

Схожесть использования знака $\to$ видна в правилах вывода:

Натуральное исчисление	λ-исчисление
$\dfrac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \cup \{x : \tau_1\} \vdash e : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . e) : (\tau_1 \to \tau_2)}$
$\dfrac{\Gamma \vdash A,\ \Gamma \vdash A \to B}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash e_1 : \tau_1,\ \Gamma \vdash e_2 : \tau_1 \to \tau_2}{\Gamma \vdash (e_2\ e_1) : \tau_2}$

Если переосмыслить суждение о типе «$t : \tau$» как утверждение о том, что t является решением задачи τ в семантике К.-Б.-Г., то правила вывода λ-исчисления переосмысливаются как правила вывода натурального исчисления для интуиционистской логики

Такую схожесть между λ-исчислением и натуральным (соответствие элементов языков, позволяющее переосмыслить λ-исчисление как интуиционистское натуральное) можно сформулировать не только для $\rightarrow$:

- $\&$ $\leftrightarrow$ тип-произведение ($\times$)
- $\vee$ $\leftrightarrow$ тип-сумма ($\cup$, $+$)
- $\mathbb{t}$ $\leftrightarrow$ единичный тип ($\top$)
- $\mathbb{f}$ $\leftrightarrow$ пустой тип ($\emptyset$, $\perp$)
- $\forall$ $\leftrightarrow$ *зависимое произведение типов*
- $\exists$ $\leftrightarrow$ *зависимая сумма типов*
- Формула $\leftrightarrow$ тип, заданный соответствием для операций
- Левая часть секвенции $\leftrightarrow$ типы аргументов/элементов программы
- Доказательство $\leftrightarrow$ терм (программа) соответствующего типа
- Доказуемость $\leftrightarrow$ непустота (*обитаемость*, *населённость*) типа

Это соответствие отражается и в правилах вывода (*но не будем перегружать рассказ*) и позволяет отождествить построение конструктивного доказательства заданного утверждения и разработку функциональной программы, имеющей заданный тип

Например, можно привести такую иллюстрацию изоморфизма для языка Haskell:

$A \to A$ — это утверждение (*логики высказываний*) доказывается программой

```
f :: A -> A
f x = x
```

$A \to B \to (A \,\&\, B)$ — это утверждение доказывается программой

```
f :: A -> B -> (A, B)
f x y = (x, y)
```

$A \,\&\, B \to A$ — это утверждение доказывается программой

```
f :: (A, B) -> A
f (x, y) = x
```

$A \to A \,\&\, B$ — а это утверждение доказать программой нельзя

И это не просто «игра словами» — например, на таком соответствии основываются популярные средства конструктивного доказательства теорем (Coq, ACL2)