Программирование с зависимыми типами

Эридан Доморацкий
(на основе курса Валерия Исаева)

6 сентября 2025 г.

Мотивация

- Типизация в языках программирования позволяет выражать свойства программ
- Чем мощнее система типов, тем больше свойств она позволяет выразить
- Зависимые типы позволяют полностью описывать спецификацию программы

Пример

- В простых системах типов мы можем приписать функции сортировки такой тип:

$$\texttt{sort} : \forall\alpha.\ \texttt{List}\ \alpha \to \texttt{List}\ \alpha$$

- Этот тип ничего не говорит о результате работы функции, кроме того, что это список элементов того же типа, что и исходный
- В языке с зависимыми типами мы можем уточнить её тип до:

$$\texttt{sort} : \forall\alpha.\ \texttt{List}\ \alpha \to \texttt{SortedList}\ \alpha$$

- И даже так мы всё ещё не полностью описали её...
  Что осталось за кадром?

Пример

- В простых системах типов мы можем приписать функции сортировки такой тип:

$$\texttt{sort} : \forall \alpha.\ \texttt{List}\ \alpha \to \texttt{List}\ \alpha$$

- Этот тип ничего не говорит о результате работы функции, кроме того, что это список элементов того же типа, что и исходный
- В языке с зависимыми типами мы можем уточнить её тип до:

$$\texttt{sort} : \forall \alpha.\ \texttt{List}\ \alpha \to \texttt{SortedList}\ \alpha$$

- И даже так мы всё ещё не полностью описали её...
  Что осталось за кадром? Результирующий список должен содержать те же элементы, что и исходный

Альтернативы

- Если мы хотим описать тип отсортированных списков, то нам нужно уметь выражать произвольные логические формулы
- Тогда мы можем определить этот тип следующим способом:

$$\{\texttt{xs} : \texttt{List}\ \alpha \mid \forall i, 2 \leq i \leq \texttt{length xs} \rightarrow \texttt{xs[i - 1]} \leq \texttt{xs[i]}\}$$

- Если мы хотим реализовать функцию `sort`, то нам нужно не только уметь выражать утверждения, но и их доказательства
- Тогда мы могли бы разделить язык на две части: отдельно программы и отдельно доказательства

Соответствие Карри-Говарда

- Зависимые типы предоставляют более гибкий и удобный подход
- Логические формулы можно записывать на языке типов, тогда доказательство — это программа соответствующего типа:

Логическая связка	$\bot$	$\top$	$\to$	$\wedge$	$\vee$
Конструктор типа	`Void`	`()`	`->`	`(,)`	`Either`

- Благодаря этому нет необходимости в двух разных языках

Зависимые типы

- При помощи простых типов можно выражать только формулы пропозициональной логики:

$$((P \to Q) \to P) \to P \quad \simeq \quad ((\alpha \to \beta) \to \alpha) \to \alpha$$

- Для формулировки интересных утверждений нам нужны кванторы
- Аналогами кванторов являются зависимые типы

Лямбда-куб

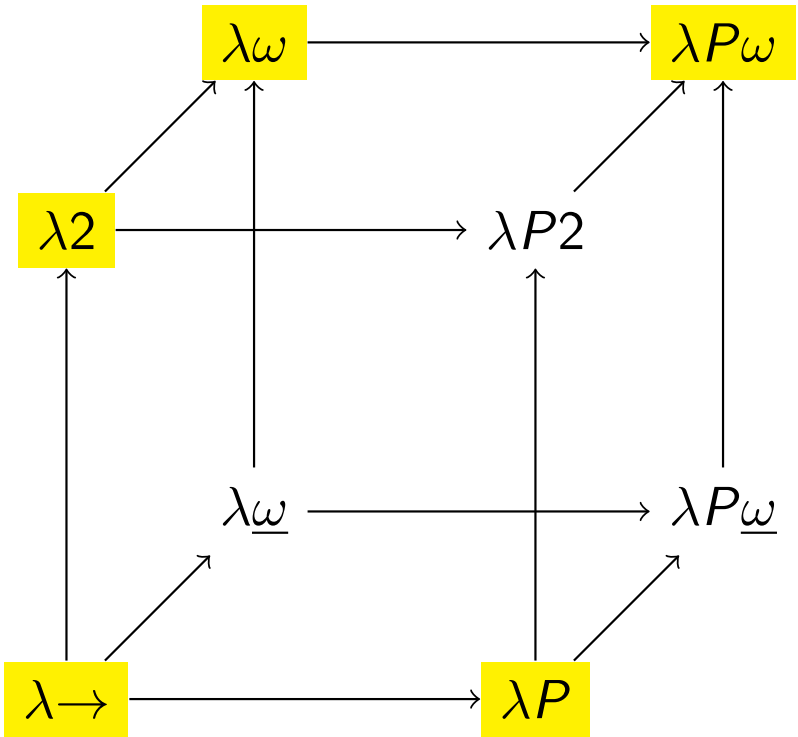

Рис.: Лямбда-куб

- $\lambda\!\rightarrow$ — просто-типизированное λ-исчисление, STLC
- $\lambda 2$ — System F, Haskell
- $\lambda\omega$ — GHC Haskell
- λP — STLC с зависимыми типами
- $\lambda P\omega$ — исчисление конструкций, Coq, Agda

Алгебра типов

- Мы умеем думать о типах с точки зрения количества элементов:
  - `Void` — пустой
  - `()` — один элемент
  - `Either a b` — $a + b$ элементов
  - `(a, b)` — $a \cdot b$ элементов
  - `a -> b` — b^a элементов
- Зависимые типы привносят в эту схему дополнительную степень свободы

Σ-типы

- Как закодировать квантор существования в типах?

Σ-типы

- Как закодировать квантор существования в типах?
- Квантор существования можно представлять как дизъюнкцию:

$$\exists x \in A.\ P(x) \simeq P(x_1) \vee P(x_2) \vee ... \vee P(x_n)$$

Σ-типы

- Как закодировать квантор существования в типах?
- Квантор существования можно представлять как дизъюнкцию:

$$\exists x \in A.\ P(x) \simeq P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)$$

- Дизъюнкцию мы кодируем как тип-сумму:

$$P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n) \simeq P\ x_1 + P\ x_2 + \ldots + P\ x_n$$

Σ-типы

- Как закодировать квантор существования в типах?
- Квантор существования можно представлять как дизъюнкцию:

$$\exists x \in A.\ P(x) \simeq P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)$$

- Дизъюнкцию мы кодируем как тип-сумму:

$$P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n) \simeq P\ x_1 + P\ x_2 + \ldots + P\ x_n$$

- Обобщим тип-сумму до Σ-типа:

$$P\ x_1 + P\ x_2 + \ldots + P\ x_n \simeq \sum_{x:A} P\ x$$

- Значениями Σ-типов являются зависимые пары:
если $a : \mathtt{A}$ и $b : \mathtt{B}\ \mathtt{a}$, то $(a, b) : \sum_{a:\mathtt{A}} \mathtt{B}\ \mathtt{a}$, то есть тип второго элемента зависимой пары зависит от значения первого элемента
- С другой стороны, Σ-типы обобщают типы-произведения до типов зависимого произведения
- Таким образом, тип независимого произведения $A \times B$ может быть записан как $\sum_{a:A} B$, где B не зависит от a

П-типы

- Как закодировать квантор всеобщности в типах?

П-типы

- Как закодировать квантор всеобщности в типах?
- Квантор всеобщности можно представлять как конъюнкцию:

$$\forall x \in A.\ P(x) \simeq P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n)$$

П-типы

- Как закодировать квантор всеобщности в типах?
- Квантор всеобщности можно представлять как конъюнкцию:

$$\forall x \in A.\ P(x) \simeq P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n)$$

- Конъюнкцию мы кодируем как тип-произведение:

$$P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n) \simeq P\ x_1 \times P\ x_2 \times ... \times P\ x_n$$

П-типы

- Как закодировать квантор всеобщности в типах?
- Квантор всеобщности можно представлять как конъюнкцию:

$$\forall x \in A.\ P(x) \simeq P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n)$$

- Конъюнкцию мы кодируем как тип-произведение:

$$P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n) \simeq P\ x_1 \times P\ x_2 \times ... \times P\ x_n$$

- Обобщим тип-произведение до П-типа:

$$P\ x_1 \times P\ x_2 \times ... \times P\ x_n \simeq \prod_{x:A} P\ x$$

- Значениями П-типов являются зависимые функции: у которых тип возвращаемого значения зависит от значения переданного аргумента
- С другой стороны П-типы обобщают стрелочные типы (типы-экспоненты) до типов зависимых функций
- Таким образом, тип независимой функции $A \to B$ может быть записан как $\prod_{a:A} B$, где B не зависит от a

Применения

Языки с зависимыми типами используют для двух разных целей:

- Во-первых, для верификации программ:
  - Мы можем записывать формальные свойства алгоритмов на языке типов, после чего доказывать их выполнение
  - Некоторые языки с зависимыми типами предоставляют возможность экстракции программ в исполняемые языки, чтобы минимизировать человеческий фактор
- Во-вторых, для формализации математики:
  - Раз уж языки с зависимыми типами позволяют нам записывать произвольные логические формулы, мы можем записывать на них и абстрактные математические утверждения и доказывать их в выбранной аксиоматике
  - Языки программирования предоставляют некоторую аксиоматику «из коробки» и зачастую позволяют расширять её дополнительными аксиомами

Реализации

- Существует несколько языков с зависимыми типами: Agda, Idris, Rocq (Coq), Lean, F* и т. д.
- Мы будем использовать Arend:
  - Использует преимущественно λ-синтаксис (в отличие от Coq и ему подобных), что позволит смотреть наиболее конкретно на суть происходящего
  - Использует гомотопическую теорию типов (HoTT), что позволит нам посмотреть на богатство и гибкость современных зависимых систем типов (чтобы другие потом казались убогими)
  - Интегрирован в IntelliJ IDEA, что упрощает начало работы с языком (потому что не надо разбираться с установкой инструментария для программирования)